Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

Análisis de unidades inerciales de medida (IMU) y diseño de controlador de ángulo de ataque aplicado a cuadricóptero

Autor: Sergio Martín Gómez

Tutor: D. Iván García Daza

2015

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

Análisis de unidades inerciales de medida (IMU) y diseño de controlador de ángulo de ataque aplicado a cuadricóptero

Autor: Sergio Martín Gómez

Director: D. Iván García Daza

Tribunal:

Presidente: D. David Fernández Llorca

Vocal 1º: D. Ignacio Parra Alonso

Vocal 2º: D. Iván García Daza

Calificación: .....

Fecha: .....

Agradecimientos

Este trabajo es el fruto de muchas horas de estudio y de trabajo que no habría sido posible sin la ayuda de mis compañeros Alejandro y Marcial y de mi director de proyecto Iván García.

Agradezco también a mi padre y a mi hermano su apoyo incondicional a lo largo de toda la carrera, que ha sido fundamental para superarla.

Índice general

Índice de figuras

Índice de tablas

Índice de listados de código fuente

Resumen

Este Trabajo de Fin de Grado ha consistido en estimar los ángulos de inclinación de una plataforma, sobre la que se han colocado dos motores, con el fin de estabilizarla en un ángulo de referencia indicado por el usuario. Para la estimación de los ángulos se ha utilizado un conjunto de sensores llamados giróscopos y acelerómetros, ambos de tres ejes, que están integrados en un sensor llamado IMU. Los motores se han controlado mediante un ESC (Electronic Speed Controller), que además cumple la función de inversor.

Palabras clave: IMU, ESC, Arduino, Giróscopo, Acelerómetro .

Abstract

This Final Year Project's purpose is to estimate the tilt angle of a platform, which has two engines installed above, in order of stabilize it in the angle that the user desires. To estimate the angles a set of sensors called gyroscopes and accelerometers have been used, both of three axis, integrated into a sensor called IMU. The engines have been controlled using a electronic speed controller (ESC), that also serves as a DC/AC converter.

Keywords: IMU, ESC, Arduino, Gyroscope, Accelerometer.

Resumen Extendido

Este proyecto consiste en controlar los dos motores que se encuentran en un eje de un cuadridóptero para que se ajuste a un ángulo de referencia pasado como entrada al sistema. El proyecto consta de varias etapas que se explicarán brevemente en este resumen.

La primera etapa de este proyecto será estudiar los componentes disponibles de los distintos fabricantes y decidir cuáles son los más adecuados para el proyecto en función de su precisión, de su facilidad de uso y del precio. Los componentes necesarios para llevar a cabo este proyecto son:

- **Sensor de posición**
- **Microcontrolador**
- **Controladores de los Motores (ESC)**
- **Motores**
- **Batería o fuente de alimentación**

Una vez elegidos los componentes que se van a utilizar se debe realizar un estudio del principio de medida de los sensores utilizados. Los sensores que se van a utilizar para estimar el ángulo de la plataforma, ambos integrados en una IMU (Inertial Measurement Unit), son:

- **Giróscopo**
- **Acelerómetro**

El principio de medida de estos dos sensores no es el mismo y en ninguno de los dos casos proporcionan una medida directa del ángulo, pero con unas sencillas operaciones se puede obtener de ambos sensores una medida de la posición. Ambos sensores presentan errores que son intolerables para esta aplicación, sin embargo utilizando los dos en conjunto se obtiene una medida de alta precisión del ángulo. Se especificará en la memoria cómo se obtiene el ángulo a partir de sus medidas.

El controlador que se va utilizar para el proyecto será un Arduino Uno, elegido por su bajo coste y la facilidad en la programación. La plataforma Arduino tiene una gran cantidad de librerías disponibles que hacen que sea muy fácil interactuar con sensores y con actuadores conectados al Arduino. La comunicación entre la IMU y el Arduino se realiza mediante el protocolo de comunicación serie I^2C, mediante el cual se pueden conectar distintos componentes al Arduino y controlarlos mediante el puerto serie. Por tanto el primer paso para la adquisición de datos será realizar un programa para la recepción de datos

mediante el puerto serie desde la IMU. Teniendo el programa de recepción de datos hecho sólo queda realizar un programa para procesar esas medidas y obtener una medida del ángulo dar finalizada esta etapa..

La siguiente etapa del proyecto consistirá en realizar un estudio del ESC (Electronic Speed Controller) y de los motores. Se debe determinar el funcionamiento de ambos componentes para poder realizar un controlador adecuado. Para el estudio se utilizarán las especificaciones del fabricante además de realizar pruebas experimentales con el fin de comprobar su funcionamiento real.

La etapa final del proyecto es la estabilización del brazo. Llegado a este punto ya se ha implementado el programa para medir el ángulo del brazo con una precisión adecuada y ya se ha estudiado la forma de variar la velocidad de los motores, por lo tanto se realizará el control del brazo mediante distintos reguladores.

CAPITULO I

Introducción

Capítulo 1

Introducción

1.1 Presentación

Para conocer el primer concepto de cuadricóptero construido hay que remontarse al año 1922. Ese año el norteamericano George de Bothezat fue el primero en hacer volar un aparato con cuatro rotores pero sin mucho éxito, pues el aparato no consiguió subir más de 5 metros.

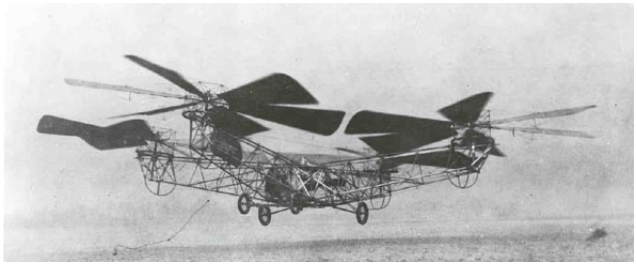


Figura 1.1: Cuadricóptero de Bothezat

Un europeo, el francés Étienne Oehmichen realizó experimentos durante los años 1923 y 1924 en los que consiguió un vuelo estacionario de cinco minutos de duración y un vuelo estacionario de siete minutos de duración elevando el aparato 10 metros sobre el suelo.

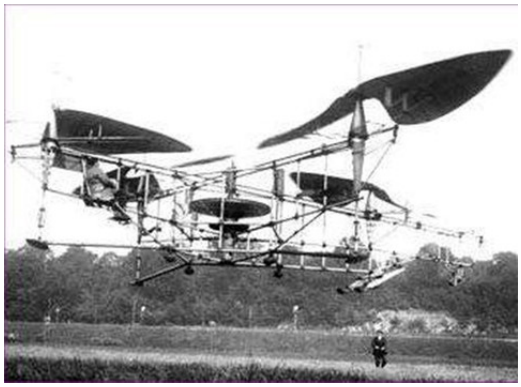


Figura 1.2: Cuadricóptero de Oehmichen

El trabajo de estos dos ingenieros se puede considerar el comienzo de los aparatos aéreos con cuatro rotores.

Debido a la complejidad en el control de estos aparatos las investigaciones se centraron más en los helicópteros de una hélice que conocemos hoy en día. Sin embargo, a lo largo de la última década el uso de aparatos aéreos de reducidas dimensiones, como cuadricópteros y drones, se ha incrementado exponencialmente. En la actualidad es muy común su uso en diversos ámbitos, siendo los más destacables su uso militar, uso comercial y uso como método de grabación en la industria cinematográfica. En todos los casos el aparato más usado es un vehículo aéreo no tripulado (UAV) basado en un sistema de aterrizaje y despegue vertical (VTOL), categoría que engloba a los cuadricópteros. Las ventajas comunes a todos los ámbitos son su facilidad de uso, facilidad de transporte y su reducido coste. Los usos que se le dan en cada campo son:

- **Militar**: se utiliza como observador de la zona en la que van a operar los soldados sin poner en riesgo vidas humanas.
- **Comercial**: varias empresas, entre las que se incluye Amazon, están investigando su uso como medio para repartir paquetes en áreas urbanas.

Figura 1.3: Cuadricóptero de Amazon en pruebas

- **Industria cinematográfica**: su uso permite reducir los costes de producción en tomas aéreas. Las tomas que hasta ahora se tenían que realizar mediante un helicóptero se pueden realizar con los cuadricópteros a un coste mucho menor.

Figura 1.4: Cuadricóptero con cámara acoplada

Estas son las aplicaciones más conocidas de estos aparatos. No obstante, tienen otras muchas aplicaciones como revisión del estado de las líneas de alta y media tensión por parte de las compañías

eléctricas, revisión del estado de los aerogeneradores mediante un modelo llamado Aracnocopter, análisis del ambiente en zonas de alto riesgo para los humanos como zonas volcánicas...

1.2 Estado del arte

En este apartado se van a enumerar algunos trabajos de otros grupos de investigación el el campo de los cuadricópteros que sean innovadores respecto a la mayoría de las investigaciones.

- En [1] se realiza el diseño y el control de un cuadricópetro con vuelo autónomo y con variador del ángulo de inclinación de las hélices. Los aspectos más destacables de este proyecto son los servos añadidos a las hélices para variar el ángulo de éstas; además ha implementado un software que permite el vuelo invertido del aparato.

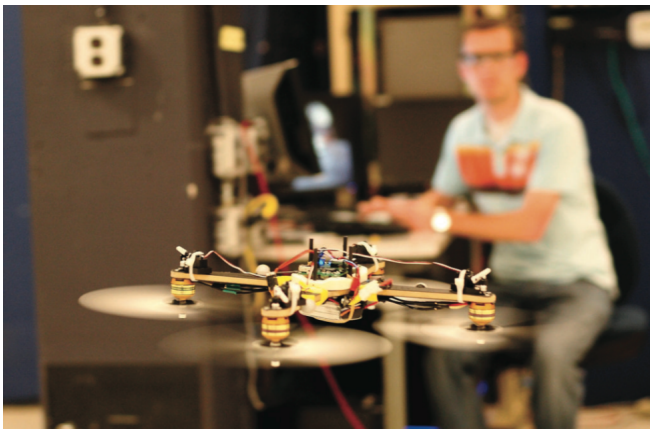


Figura 1.5: Detalle del vuelo invertido del cuadricóptero

- En [2] se utiliza una tecnología denominada BCI[1], que es una tecnología que consiste en la adquisición de ondas cerebrales para luego ser procesadas e interpretadas por un ordenador. En él se realiza un experimento en el que se controla un cuadricópetro en un espacio tridimensional usando un electroencefalograma no invasivo.

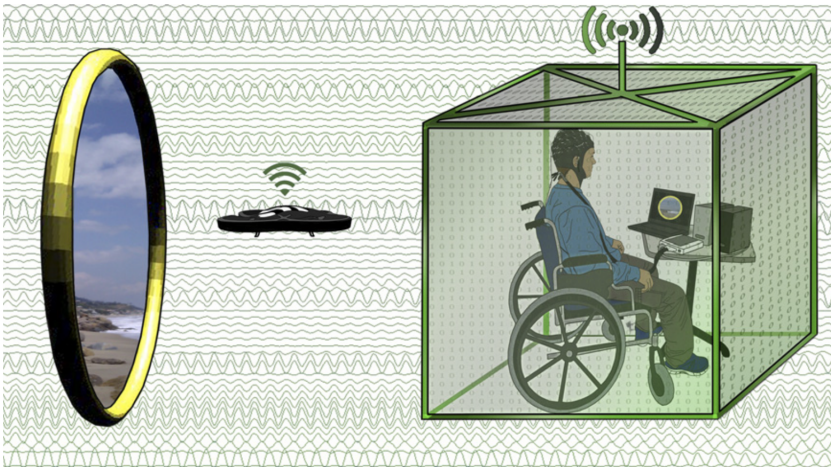


Figura 1.6: Diagrama conceptual de la potencial función de la robótica de telepresencia BCI impulsada en la restauración de la autonomía a una persona paralizada.

[1] Siglas de Brain-computer interface (Interfaz cerebro-computadora)

1.3 Motivación y objetivos del proyecto

1.3.1 Motivación

Debido a la mayor presencia de estos aparatos año tras año en múltiples aplicaciones este proyecto se va a centrar en estudiar el método de medida de la posición de un eje de un cuadricóptero y su posterior estabilización mediante distintos reguladores. Ya que la distribución de los motores en el cuadricóptero es simétrica, el algoritmo de control de estabilización del brazo con el que se va a trabajar se puede aplicar al otro brazo para estabilizar el vuelo del cuadricóptero; por lo tanto este proyecto no se va a centrar en aspectos mecánicos ni en la aerodinámica del aparato sino en la teoría de control y en la programación del controlador para procesar los datos de los sensores y usarlos para el algoritmo de estabilización.

Mediante la realización de este proyecto se afianzan conocimientos estudiados durante la carrera en los campos de:

- **Control**: se deben utilizar los conocimientos adquiridos en las distintas asignaturas de control para el control del brazo.
- **Circuitos eléctricos**: se deben conectar dos circuitos muy simples, uno de control que enviará las señales de comunicación entre sensor y Arduino y entre ESC y Arduino, y otro circuito de potencia que alimentará los motores desde la fuente de alimentación que se utilice.
- **Programación**: se requieren conocimientos estudiados en asignaturas de programación y en sistemas digitales (para el manejo de registros de un microcontrolador) para programar el microcontrolador, que generalmente requieren el lenguaje C o C++.
- **Instrumentación Electrónica**: se utilizará el programa LabView aprendido en esta asignatura para realizar una interfaz gráfica que se comunique con el microcontrolador y permita controlar el brazo con más facilidad.

1.3.2 Objetivos

El proyecto se dará por concluido cuando se cumplan los siguientes objetivos:

1. Análisis y estimación de de los 3 grados de libertad del cuadricóptero obtenidos de la IMU.
2. Diseño del PID y de otros reguladores para el control del ángulo del brazo.
3. Programación de la interfaz gráfica en LabView para manejar el brazo.

1.4 Medios y herramientas necesarios

En este apartado sólo se enumerarán los recursos necesarios para la realización del proyecto, más adelante, en otro apartado, se explicarán más en detalle las características de cada uno de los componentes y programas aquí enumerados. Para realizar este proyecto es necesario disponer de:

- Microcontrolador
- IMU
- LabView
- PC

- Batería o fuente de alimentación
- Motores Brushless
- Hélices
- Controladores de velocidad (ESC)
- Estructura de ensamblaje
- Matlab
- AutoDesk Inventor

CAPITULO II

Elección de los componentes

Capítulo 2

Elección de los componentes

En este apartado del trabajo se van a a estudiar los distintos componentes presentes en el mercado y se van a elegir los más adecuados. De cada componente se comentarán sus especificaciones y sus características detalladamente.

2.1 Controlador

Este componente es el cerebro del cuadricóptero, es el encargado de recoger todos los datos de los sensores para procesarlos y generar las señales necesarias que se enviarán a los motores para estabilizar el brazo. Para esta aplicación no se requiere un procesador con mucha capacidad de cómputo, los requisitos necesarios para que el controlador sea válido son:

1. **Puerto serie**: es necesario para la comunicación con el sensor y con el ordenador. Mediante la lectura del puerto serie se leerán los datos de los sensores y mediante la escritura en el puerto serie se enviarán órdenes desde el programa realizado en el ordenador al controlador.
2. **Entradas/Salidas digitales con PWM**: la excitación de los motores se realiza mediante una señal PWM, la cual se envía a través de estos puertos.

Con estas características se puede buscar en Internet la oferta de controladores disponible que cumplan las características especificadas. Los dos controladores más completos y con mejor relación calidad/precio son el Arduino Uno y el A-Star 32U4 Prime SV, cuyas características se enumerarán a continuación.

2.1.1 Arduino Uno

Esta placa está basada en un microcontrolador ATmega328 y dispone de 14 entradas/salidas digitales, de las cuales 6 pueden ser usadas como salidas PWM. Se conecta al ordenador mediante USB, que sustituye al antiguo puerto serie RS232 y dispone de puertos para la comunicación serie. Por lo tanto este microcontrolador cumple con los requisitos necesarios para poder realizar este trabajo.

Una de las ventajas de usar la plataforma Arduino es la enorme cantidad de documentación disponible en Internet para cualquier proyecto relacionado con Arduino. Ya que es una plataforma de código abierto hay una comunidad de desarrolladores que ha generado toda esa información y resulta muy sencillo aprender a programar el micro.

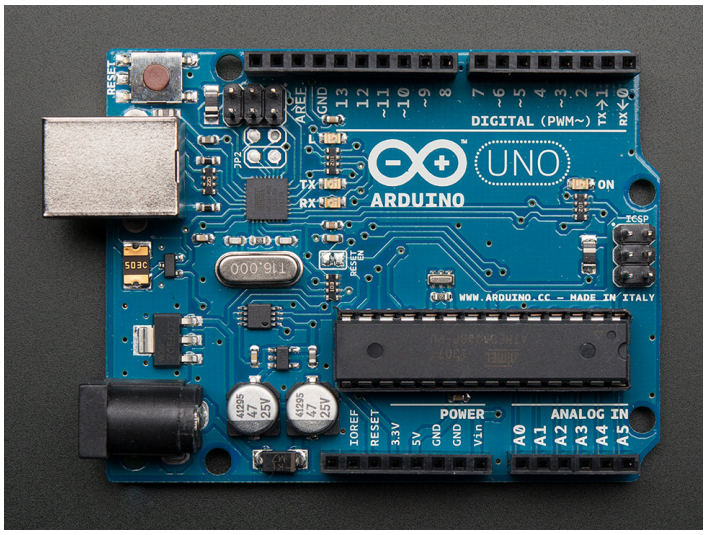


Figura 2.1: Microcontrolador Arduino Uno

2.1.2 A-Star 32U4 Prime SV

Esta placa está basada en un microcontrolador ATmega32U4 y dispone de 14 entradas/salidas digitales, de las cuales 7 pueden ser usadas como salidas PWM. Se conecta al ordenador mediante micro-USB, que sustituye al antiguo puerto serie RS232 y dispone de puertos para la comunicación serie. Por lo tanto este microcontrolador cumple con los requisitos necesarios para poder realizar este trabajo.

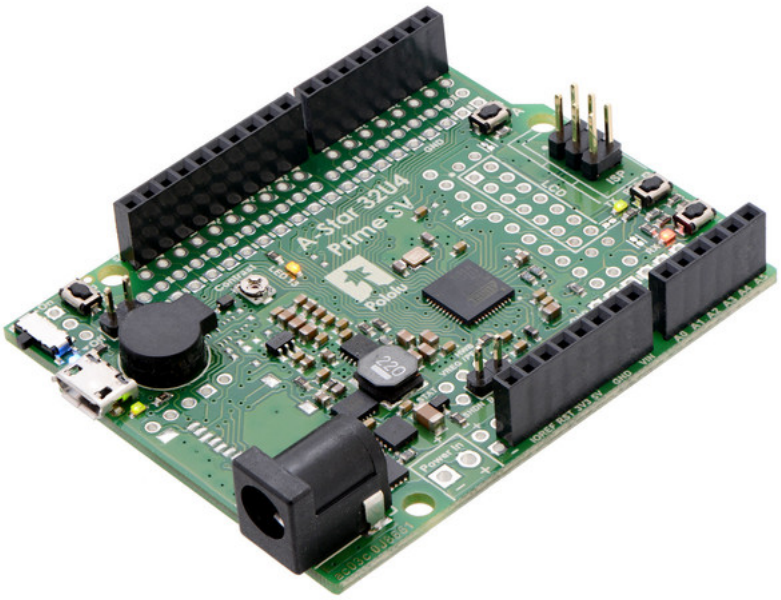


Figura 2.2: Microcontrolador A-Star 32U4 Prime SV

La ventaja de este microcontrolador respecto al Arduino es un mayor rango de voltaje de alimentación y mayor número de entrads/salidas disponibles para el usuario.

2.1.3 Comparativa y elección del microcontrolador

Tabla 2.1: Comparativa controladores programables.

	Arduino Uno	**A-Star 32U4 Prime SV**
Microcontrolador	ATmega328P	ATmega32U4
Entradas/Salidas	20	26
Salidas PWM	6	7
Rango de tensión de entrada	7 a 12 V	5 a 36 V
Precio	20 €	28 €

Las mayores diferencias entre ambos controladores están en el número de entradas y en el margen de tensiones de entrada. En este trabajo sólo se van a necesitar 2 salidas PWM, por lo tanto cualquiera de los controladores son válidos. Las tensiones de entrada tampoco son relevantes a la hora de elegir una opción u otra ya que la oferta de baterías y fuentes de alimentación es muy amplia. La mayor disponibilidad de documentación disponible en la plataforma Arduino hace que el aprendizaje y el desarrollo sean más rápidos, por lo tanto es la mejor opción. El controlador usado será el **Arduino Uno**.

2.2 Unidad inercial de medidas (IMU)

La unidad inercial de medidas o IMU por sus siglas en inglés es un dispositivo que combina un conjunto de sensores capaces de medir la aceleración, la velocidad angular y el campo magnético. Los sensores capaces de medir esas 3 magnitudes son el acelerómetro, el giróscopo y el magnetómetro. En general, las IMUs disponibles en el mercado son capaces de medir estas magnitudes en las 3 direcciones del espacio pues integran 3 acelerómetros, 3 giróscopos y 3 magnetómetros. En capítulos posteriores se explicará detalladamente el funcionamiento del giróscopo y del acelerómetro, pues el magnetómetro no es usado en este trabajo. Esta apartado se centra en comparar los modelos más apropiados para este trabajo. Las dos IMU que se van a comparar son la Alt-IMU-10 de Pololu y la MPU-6050 de Invensense.

2.2.1 AltIMU-10

Esta IMU [3] combina un barómetro digital, un giróscopo de 3 ejes, un acelerómetro de 3 ejes, un magnetómetro de 3 ejes y un altímetro. Por lo tanto es un dispositivo con 10 grados de libertad. La ventaja de elegir esta IMU es que el vendedor proporciona unas librerías mediante las que leer los valores de los sensores sin tener que recurrir a lecturas de registros; sólo hay que hacer llamadas a las funciones definidas en la librería y se leerán los datos de los sensores.

Esta IMU es compatible con el controlador elegido pues rango de tensiones de alimentación es 2.5 V a 5.5 V y la corriente necesaria 6 mA (cada pin del Arduino entrega hasta 40 mA de corriente). Además incluye un regulador de tensión de 3.3 V, cuando al pin de alimentación VIN se le alimenta a 3.3 V o más, el pin VDD actúa como una fuente de 3.3 V que puede suministrar hasta 150 mA a componentes externos. Los rangos de FS[1] se especificarán más adelante en la tabla comparativa.

[1] FS hace referencia a fondo de escala

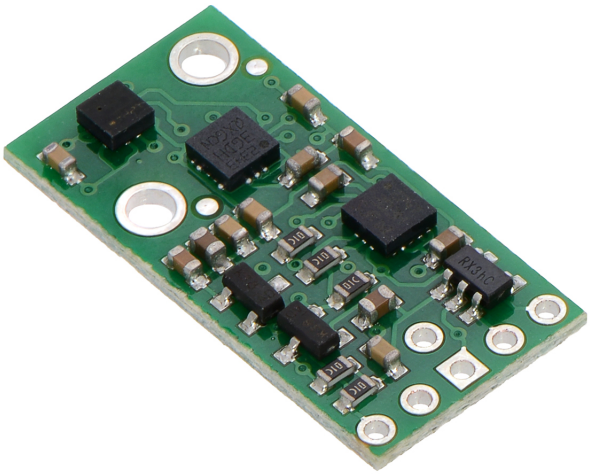


Figura 2.3: AltIMU-10

La lectura se realiza mediante comunicación I^2C, que es un protocolo de comunicación serie. Tiene un conversor digital-analógico por canal, es decir uno para cada eje de cada sensor, característica que asegura medidas de mucha precisión.

2.2.2 MPU 6050

Este dispositivo [4] combina un giróscopo de 3 ejes y un acelerómetro de 3 ejes. Es un dispositivo con 6 grados de libertad. En este caso el fabricante no proporciona librerías y las operaciones para leer datos de los sensores son algo más complejas, necesitando acceder a registros y realizar operaciones de desplazamiento de bits para obtener los datos de los sensores.

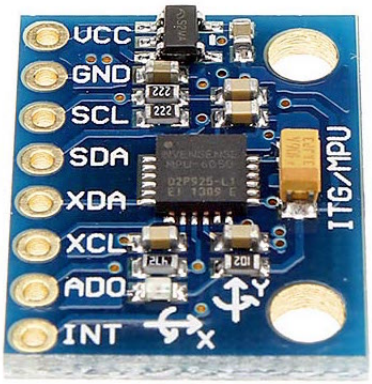


Figura 2.4: MPU 6050

También es compatible con Arduino pues su rengo de tensión de alimentación es 2.375 V a 3.46 V. La lectura se realiza mediante comunicación I^2C. Tiene un conversor digital-analógico por canal, es decir uno para cada eje de cada sensor, característica que asegura medidas de mucha precisión.

2.2.3 Comparativa y elección de la IMU

Tabla 2.2: Comparativa IMU.

	AltIMU-10	**MPU 6050**
Rangos FS giróscopos	$\pm 245, \pm 500, \pm 2000^o/s$	$\pm 250, \pm 500, \pm 1000, \pm 2000^o/s$
Rangos FS acelerómetros	$\pm 2, \pm 4, \pm 6, \pm 8, \pm 16g$	$\pm 2, \pm 4, \pm 6, \pm 8, \pm 16g$
Rango de tensión de entrada	2.5 a 5.5 V	2.375 a 3.46 V
Precio	30 €	6 €

La única diferencia significativa es el precio. No obstante la mayor facilidad a la hora de programar para leer los valores de los sensores va a reducir el error significativamente y merece la pena esa diferencia de costes. La IMU que se usará en este trabajo es la **AltIMU-10**.

2.3 Motores

La elección de unos motores adecuados resulta imprescindible, pues de sus características dependerá la estabilización y el consumo de energía. Los dos tipos de motores entre los que hay que elegir son los motores con escobillas, que generalmente son motores de continua, o motores sin escobillas. Para analizar las diferencias entre los motores se ha utilizado información obtenida de [5].

2.3.1 Motores DC (Brushed Motors)

Estos motores están compuestos de dos partes, un estator que sirve como soporte mecánico y en el que se encuentran los polos, y un rotor que recibe la corriente de las escobillas. La función de las escobillas es la de cambiar la dirección de la corriente que circula por sus bobinas, esta conmutación produce un par de rotación en el eje. Estos motores no necesitan ningún tipo de controlador para variar su velocidad debido a la conmutación mecánica mediante escobillas, por lo que el control es simple.

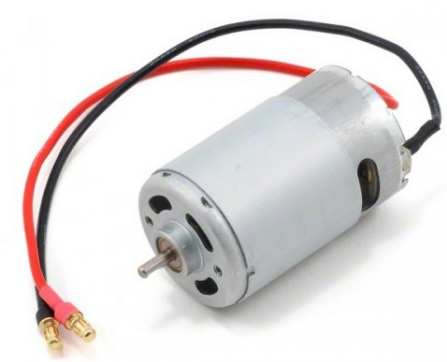


Figura 2.5: Motor de corriente continua

El principal inconveniente de este tipo de motores es que debido al rozamiento del rotor con las escobillas se produce un desgaste debido al que hay sustituir las escobillas con mucha frecuencia. En consecuencia el coste de mantenimiento es elevado.

2.3.2 Motores sin escobillas (Brushless Motors)

Son similares a los motores de corriente continua en su estructura pues usan bobinas e imanes para mover el eje pero su funcionamiento es muy distinto. Con motores de alterna trifásicos y debido a que no hay ningún elemento mecánico como las escobillas que inviertan el sentido de la corriente es necesario para regular la velocidad de giro un controlador electrónico de velocidad (ESC). El ESC además funciona como inversor, convirtiendo la corriente continua suministrada por las baterías a corriente alterna necesaria para alimentar el motor.

Figura 2.6: Motor sin escobillas

De este tipo de motores existen dos posibles configuraciones, de rotor externo o de rotor interno. Su nombre indica la posición de las bobinas del respecto a los imanes permanentes.

2.3.3 Comparativa y elección de los motores

Tabla 2.3: Comparativa Motores.

	Motor sin escobillas	**Motor continua**
Mantenimiento	Bajo	Elevado
Eficiencia	Alta	Media
Conmutación	Electrónica	Mecánica
Ruido electrónico	Bajo	Alto
Precio	15 €	7 €

Las ventajas de elegir un motor sin escobillas se observan en la tabla 1.3. Sin embargo esas no son todas las ventajas, al no haber escobillas no hay rozamiento y el rango de velocidad es más elevado al no tener limitación mecánica, también tienen mayor eficiencia pues la pérdida de calor es mucho menor y, como última consecuencia de no tener escobillas, el rendimiento es mayor, con la misma potencia eléctrica suministrada el motor sin escobillas adquiere mayor velocidad. Por todas estas ventajas, a pesar de la diferencia de coste, se va a utilizar el **Motor Brushless**. Por lo tanto el siguiente paso será elegir un controlador electrónico de velocidad.

Para estos tipos de motores el fabricante suele especificar, además de los parámetros comunes a los motores de continua, el factor Kv, que indica el número de revoluciones por minuto por cada voltio de tensión que se le aplica al motor. Ya que el peso que tienen que levantar los motores no es muy elevado,

no se requiere de un gran factor Kv ni de un gran par en los motores para este trabajo. En la tabla 2.4 se indican las especificaciones del motor elegido.

Tabla 2.4: Especificaciones motor.

	Motor sin escobillas A2212
Eficiencia Máxima	80 %
Kv	1000
Diámetro del eje	3.17mm
Tensión alimentación	7.4 a 14.8 V
Corriente sin carga	0.5 A
Corriente Máxima	12 A

2.4 Controlador electrónico de velocidad

Los motores brushless, como se ha mencionado en la sección 2.3, necesitan para regular su velocidad de giro mediante la conmutación de la sentido de la corriente un dispositivo llamado controlador electrónico de velocidad. Es un circuito electrónico utilizado para variar la dirección del motor, la velocidad e incluso para actuar como freno.

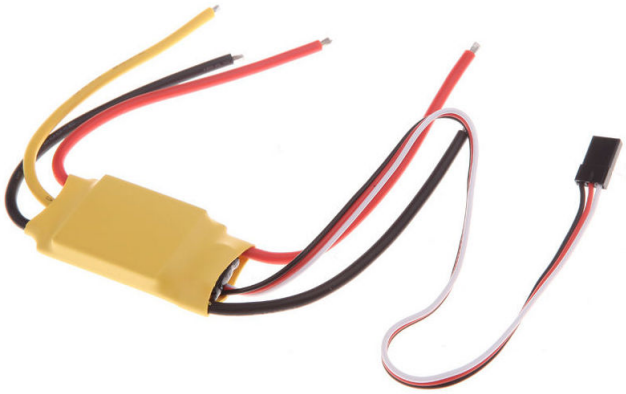


Figura 2.7: ESC.

Los distintos cables que se ven en la figura 2.7 corresponden a:

- **3 cables izquierda (Amarillo, rojo y negro)**: cables de conexión al motor, cada uno corresponde a una fase.
- **2 cables derecha (Rojo y negro)**: estos cables deben conectarse a la batería, son los cables de alimentación.
- **Conector de 3 cables derecho**: estos 3 cables son los cables de señal, el rojo y negro corresponden a Vcc y a masa y el blanco es el cable de señal.

Los controladores están regulados por una señal PPM (modulación por posición de pulso), que son similares a las señales PWM. En este tipo de señales la tensión aplicada en bornes del motor será el valor medio del tren que estará acotado entre 1 y 2 ms, es decir, con 1 ms el motor gira a la velocidad mínima

y con 2 ms el motor gira a la velocidad máxima. La frecuencia a la que se trabaja en este proyecto es de 33.33 Hz.

Un ejemplo de una señal de excitación para un ESC se muestra en la figura 2.8.

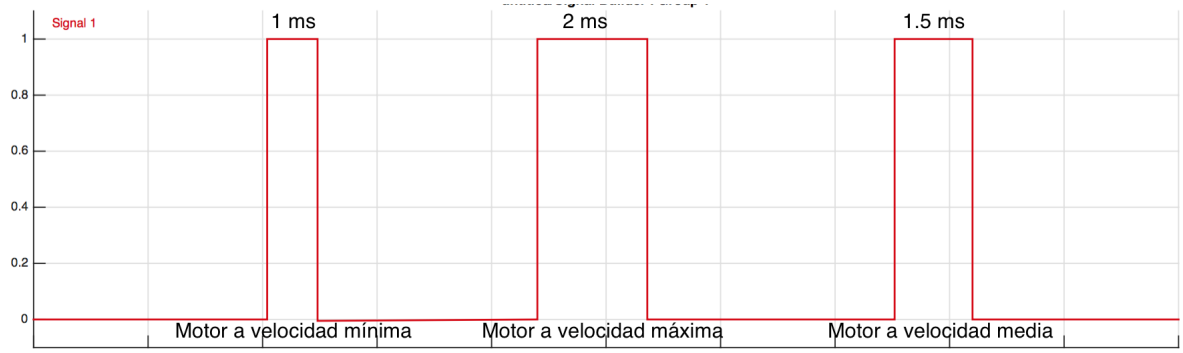


Figura 2.8: Señal ESC.

En esta señal se puede observar el rango de velocidades y el funcionamiento del motor en cada caso. Si se excita el motor a una velocidad menor de la mínima no arrancará y si se excita a una velocidad mayor de la máxima existe el riesgo de romper el motor. La forma de mandar estas señales al ESC se explicará en detalle en el capítulo IV.

Las características de las distintas opciones disponibles en el mercado son muy similares, por lo tanto se ha elegido un ESC que tiene una buena relación calidad/precio. Sus especificaciones se muestran en la tabla 2.5.

Tabla 2.5: Especificaciones ESC.

	ESC
Salida	30 A continuo, 40A hasta 10 secs
Voltaje de entrada	2-4 baterías LiPo

2.5 Hélices

Como el brazo sólo va a realizar un movimiento de rotación en un solo eje no se necesitan unas hélices muy grandes pues no es necesaria mucha fuerza de empuje. Por ello se han elegido unas hélices de unas dimensiones estándar que darán la fuerza de empuje necesaria para mover el brazo. Las dimensiones de las hélices son 200 mm de un extremo a otro y 6 mm el diámetro del eje.

2.6 Batería o fuente de alimentación

Las corrientes que necesitan los motores son bastante elevadas debido a la oposición de las bobinas a cambios en la corriente que circulan a través de ellas. Las baterías que se suelen utilizar en los cuadricópteros se denominan baterías LiPo, que son baterías de polímero de Litio. Cada una de estas baterías está compuesta por celdas en paralelo, cada celda suele tener un voltaje de 3.7 V y las ventajas que tienen es que son muy ligeras (conveniente para el cuadricóptero) y proporcionan grandes descargas de corriente.

Figura 2.9: Batería LiPo

Uno de los problemas de estas baterías es que si se descargan demasiado, es decir, la tensión es inferior a un valor facilitado por el fabricante, la batería deja de funcionar. Además si se les suministra demasiada corriente en la recarga pueden llegar a explotar. Ya que este trabajo está centrado en estabilizar un brazo que va a tener una base de apoyo se va a utilizar una fuente de alimentación, que son más seguras y la tensión proporcionada es más estable.

Figura 2.10: Fuente de alimentación utilizada

La fuente utilizada tiene dos fuentes independientes, una de 5 V y una de 12 V. En este trabajo se utilizará la fuente de 12 V. Las características de la fuente de alimentaciónse muestran en la tabla 2.6.

Tabla 2.6: Especificaciones fuente de alimentación.

	5 V	**12 V**
Salida	13 A (Pico)	6 A (Pico)
Entrada	180-240 V	180-240 V

2.7 Montaje final

Para montar el brazo se necesita un material que sea ligero y que sea fácil de trabajar, para que el montaje de la estructura no resulte costosa. El material usado en este trabajo es el aglomerado, que se obtiene a partir de virutas o serrín. Su manipulación sólo requiere de una sierra de vaivén y se puede taladrar fácilmente para acoplar las distintas partes. El montaje del brazo debe tener suficiente peso y suficiente superficie en la base para que ante movimientos bruscos en la fase de pruebas del PID la estructura no se balancee, sólo debe moverse el brazo a estabilizar.

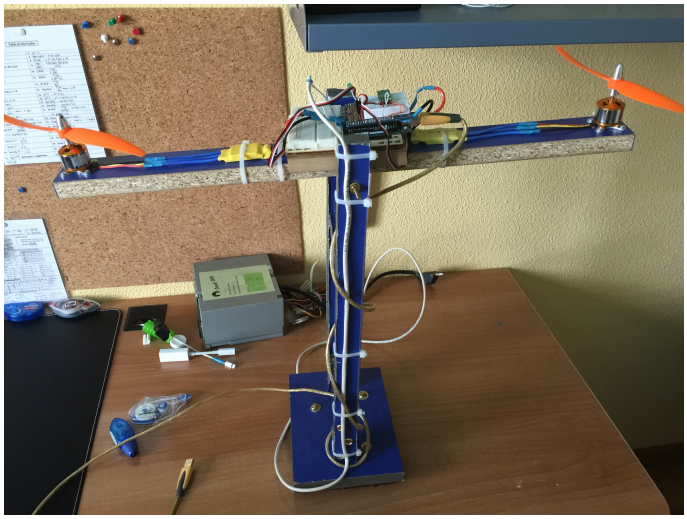
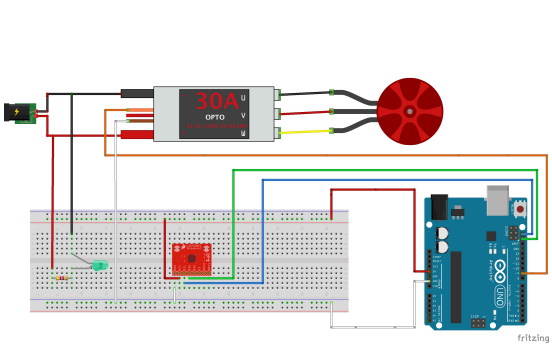
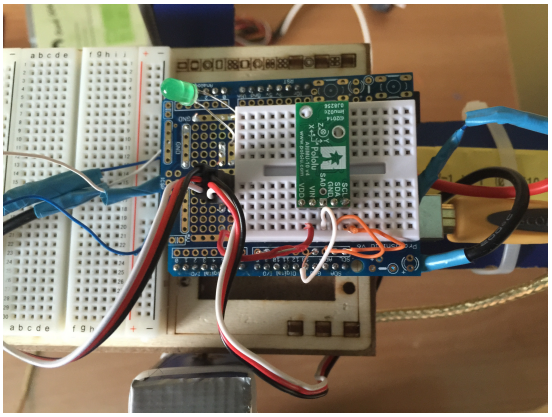


Figura 2.11: Brazo montado.

La estructura de la figura 2.11 tiene un eje central sobre el que puede girar libremente el brazo. Para que las hélices no pasen de un ángulo de aproximadamente 75º por cada lado, caso en el que chocarían contra las barras de soporte del brazo y sería peligroso, se ha colocado un tope de madera entre las barras para limitar el movimiento; además se limitará también por software para mayor seguridad. Para que el centro de gravedad del montaje descienda se ha colocado una pesa en la base y se disminuye el riesgo de volcar ante acelerones bruscos de los motores.



(a) Esquema del circuito.



(b) Foto del circuito real.

Figura 2.12: Imágenes del esquema y del circuito montado.

En la figura 2.12 se muestran el esquema del circuito montado y una imagen real del circuito. En el esquema sólo se ha representado uno de los motores porque la conexión del otro es idéntica.

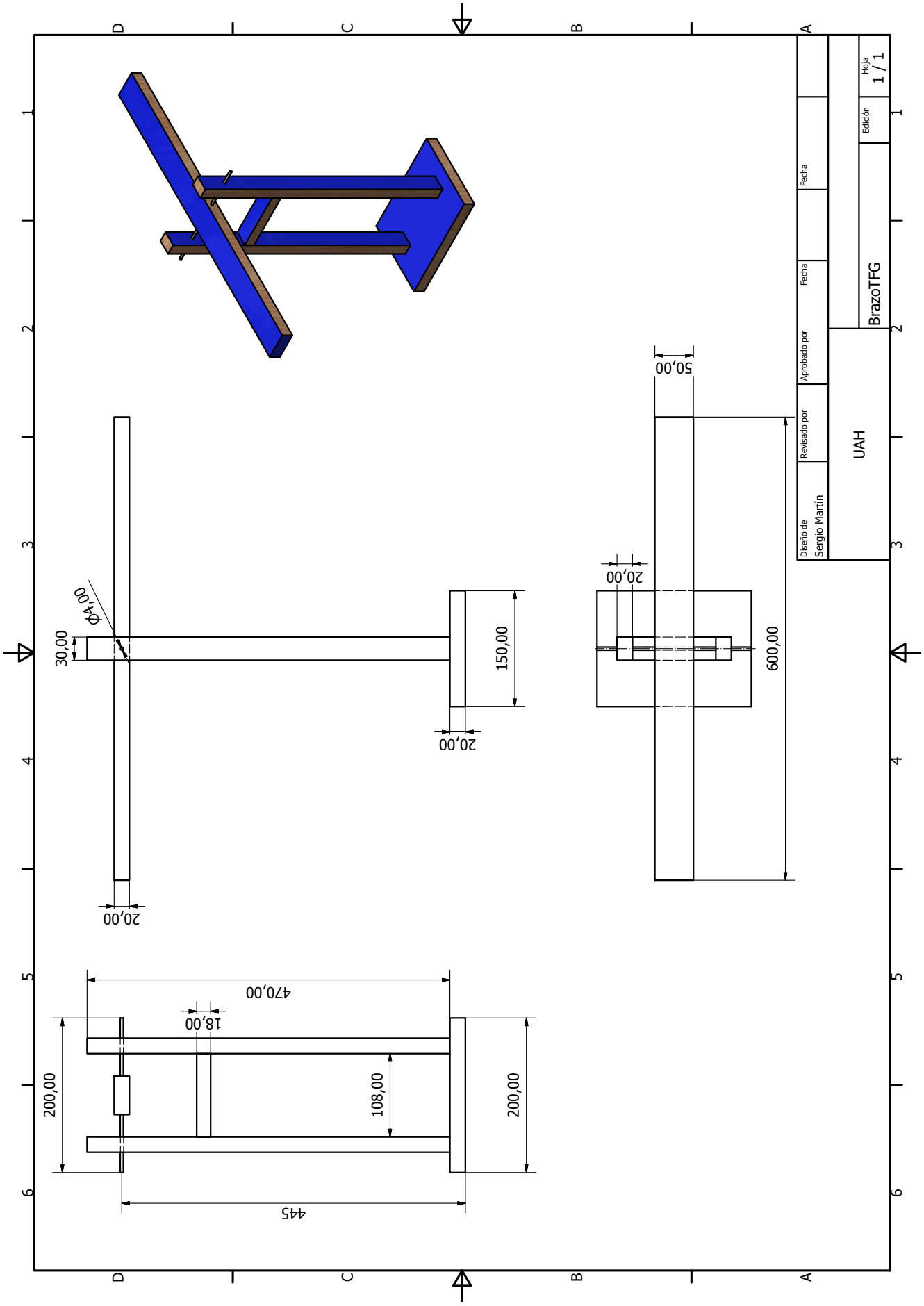


Figura 2.13: Alzado, planta y perfil de la estructura.

CAPITULO III

Sensores de posición

Capítulo 3

Sensores de posición

3.1 Introducción

Como ya se ha comentado previamente, en este trabajo se van a utilizar dos sensores para estimar el ángulo del brazo, el giróscopo y el acelerómetro. Ninguno de los dos sensores da una medida directa del ángulo, por lo tanto hay que realizar algunas cálculos para estimar el ángulo. En este capítulo se explicarán todos los pasos necesarios para llegar a obtener medidas precisas del ángulo del brazo, desde la conexión de los componentes hasta la realización de los programas.

Ambos sensores están integrados integrados en un chip denominado IMU. La IMU utilizada es la AltIMU-10 de Pololu e integra en el mismo chip un giróscopo de 3 ejes, un acelerómetro de 3 ejes, un magnetómetro de 3 ejes y un altímetro dando un total de 10 medidas independientes, lo que forma un sensor de 10 DOF[1].

3.2 Conexión de la IMU

Para poder tomar medidas de la IMU hay que conectar 4 pines, los de alimentación y los de la comunicación serie. Los pines de la IMU se muestran en la figura 3.1. Los pines de alimentación son el pin Vin para el positivo y GND para masa. Se puede conectar a un rango de tensiones de entre 2.6 y 5.5 V. La corriente de suministro que necesita es 6 mA, cada pin de Arduino suministra 40 mA, por lo que ese requisito lo cumple holgadamente. Los pines de comunicación son SCL y SDA, que corresponden a la señal de reloj y a la línea de datos, respectivamente. El valor alto corresponde a una tensión Vin y el valor bajo corresponde a 0 V.

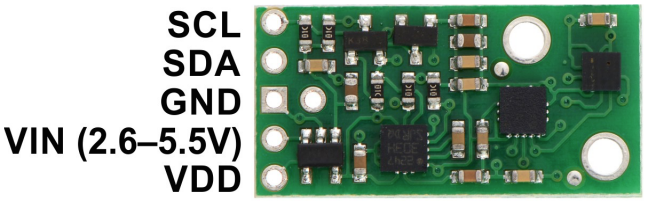


Figura 3.1: IMU con los nombres de los pines.

[1] Grados de libertad

El pin Vdd corresponde a una fuente de tensión de 3.3 V para otros componentes conectados al circuito, por lo que en ete trabajo se dejarán al aire. El pin Vdd corresponde a una fuente de tensión de 3.3 V para otros componentes conectados al circuito, por lo que en ete trabajo se dejarán al aire. En la figura 3.2 se muestra el esquema del circuito conectado.

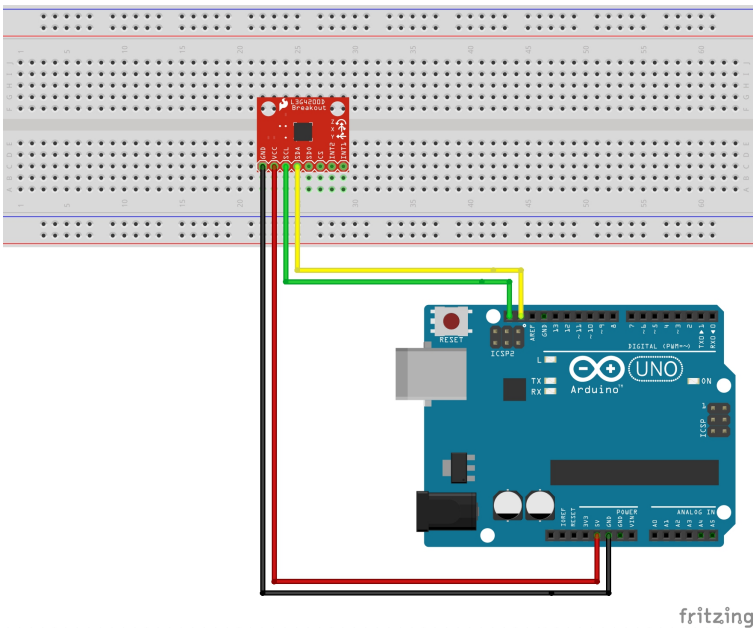


Figura 3.2: Esquema del circuito.

Uno de los aspectos a tener en cuenta para identificar correctamente el ángulo medido es el sentido de los ejes en la IMU y la dirección en que incrementan y decrementan cuando la IMU gira. Se explicará mediante la figura 3.3, que representa la conexión realizada en el brazo.

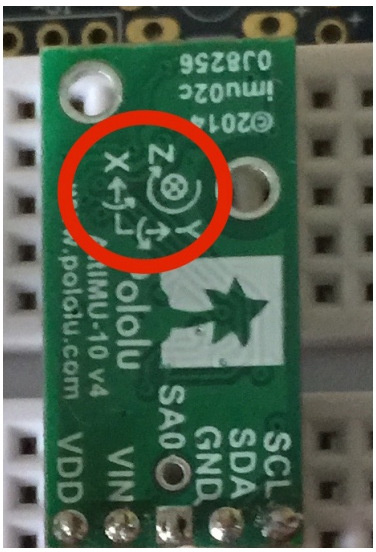


Figura 3.3: Conexión de la IMU.

Teniendo una vista global de la conexión completa en la figura 2.12, en esta vista en detalle se sabe que el eje que se va a medir es el eje X. La flecha significa que en la dirección hacia la que apunta la flecha los valores del sensor crecen y en el sentido contrario decrecen.

3.3 Giróscopo

Es un sensor que sirve para medir la velocidad angular de un algún aparato o vehículo. En la figura 3.4 se observa su estructura. Para medir la velocidad de giro utiliza los principios del momento angular. Como se indica en [6], cuando el giróscopo se somete a un momento de fuerza que tiende a cambiar la orientación de su eje de rotación, éste cambia de orientación girando respecto a un tercer eje, perpendicular tanto a aquel respecto del cual se lo ha empujado a girar, como a su eje de rotación inicial. Si está montado sobre un soporte de Cardano[2] que minimiza cualquier momento angular externo, o simplemente gira libre en el espacio, el giroscópo conserva la orientación de su eje de rotación ante fuerzas externas que tiendan a desviarlo.

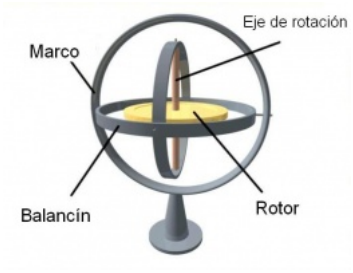


Figura 3.4: Giróscopo mecánico.

Debido a los tamaños con los que se trabaja hoy en día en el campo de la electrónica resulta imposible construir un giróscopo electrónico con la estructura de la figura 3.4, por lo que se suelen utilizar giróscopos denominados de estructura vibrante o de vibración de Coriolis. Como se indica en [7, p. 2] funcionan como resultado de fuerzas de Coriolis desarrolladas que ocurren cuando una partícula experimenta movimiento lineal en un bastidor rotatorio de referencia.

Conociendo los tipos de giróscopos y su funcionamiento ya se puede pasar a realizar el programa para tomar medidas con el giróscopo utilizado en este trabajo. El giróscopo integrado en la IMU utilizada es el L3GD20H de la marca STMicroelectronics, un giróscopo de 3 ejes con salida digital. Se comunica mediante el bus de comunicación serie I^2C. Como ya se ha comentado previamente el fabricante del giróscopo adquirido provee una librería para que la lectura de datos de los sensores de la IMU resulte más sencilla. A continuación se muestran los pasos a seguir para poder empezar a leer datos del sensor:

1. **Incluir la librería**: para poder utilizar las funciones y clases de datos definidas en la librería hay que incluir en el proyecto mediante un include.

2. **Declarar una variable**: en la librería hay definida una clase de datos llamada L3G que será la que usaremos para leer datos del giróscopo, por lo tanto hay que declarar una variable de este tipo para llamar a las funciones de la librería.

3. **Inicializar**: el giróscopo por defecto está configurado en modo "sleep"; hay que inicializarlo para poder empezar a leer datos. Además para poder comunicarse con Arduino hay que inicializar la comunicación serie.

4. **Lectura datos**: con los pasos previos ya realizados se pueden empezar a leer datos con la función *read* definida en la librería.

[2]Es un mecanismo de suspensión consistente en dos aros concéntricos cuyos ejes forman un ángulo recto.

Un ejemplo de un programa mediante el que se podrían leer datos del giróscopo se muestra en el código 3.1. En él se han programado todos los pasos arriba mencionados para tomar medidas del giróscopo.

Codigo 3.1: Ejemplo de programa para tomar medidas del giróscopo.

```
#include <Wire.h>               //Librería para la comunicación serie
#include <L3G.h>                  //Librería del giróscopo

L3G gyro;                   //Declaración variable del tipo L3G

void setup()
{
  Serial.begin(115200);             //Inicia el puerto serie
  Wire.begin();                 //Comunicación I2C entre IMU y Arduino

   if (!gyro.init())                //Inicialización del giróscopo
  {
    Serial.println("Failed to autodetect gyro type!");
    while (1);
  }

  gyro.enableDefault();//Habilita varios registros a su valor por defecto

  }

void loop()
{
  gyro.read();
}
```

Estos valores leídos del sensor y almacenados en la variable gyro son valores "raw", es decir, es un entero de 16 bits, no la velocidad angular. Para obtener la velocidad angular hay que realizar una serie de operaciones especificadas por el fabricante en la hoja de características del sensor. Para obtener valores de velocidad angular se debe multiplicar el valor "raw" obtenido por un factor de conversión llamado sensibilidad, cuyas unidades son mdps[3]/digit. Los valores de estas constantes para cada FS se encuentran en [8, p. 10]. El objetivo final no es obtener la velocidad angular sino la posición, que es el ángulo. Como la velocidad es la derivada de la posición respecto del tiempo la solución es aplicar la ecuación 3.1 para obtener la posición.

$$angulo = \int dps\_x \, dt \tag{3.1}$$

Para poder programar esa integral hay que controlar el tiempo que tarda en ejecutarse el bucle principal, que debe ser siempre el mismo. Se debe medir el tiempo al inicio del bucle y al final del bucle para asegurarse de que en ninguna iteración se sobrepasa el tiempo de ejecución y si está por debajo introducir un delay. Para medir el tiempo transcurrido se puede utilizar la función "millis()".

Codigo 3.2: Ejemplo para obtener una medida de la posición.

```
#include <Wire.h>
#include <L3G.h>

#define FACTOR_CONVERSION_245DPS 0.00875

```

[3]mdps hace referencia a milidegrees per second (miligrados por segundo)

```
#define DT 0.03               //Tiempo en segundos

int16_t dps_x;
unsigned long time_loop1, time_loop2; //Variables para medir el tiempo de ejecución
double angulo_giroscopio_x;

...

void loop()
{

  time_loop1 = millis();
  gyro.read();

  dps_x =  (gyro.g.x)*FACTOR_CONVERSION_245PS;
  angulo_giroscopio_x += (double) dps_x*DT;     //La integral se realiza mediante una
      multiplicación

  time_loop2 = millis();

  while(time_loop2-time_loop1 < DT)
  {
    delay(1);
  }
}
```

Con el código 3.2 ya se obtiene una estimación del ángulo del brazo, sin embargo esa medida tiene fuentes de error que hacen que la precisión de la medida sea muy baja. Hay métodos para minimizar estos errores pero no se pueden eliminar completamente. A continuación se comentan estos errores y cómo reducirlos.

3.3.1 Fuentes de error

3.3.1.1 Offset

Este error se refiere al valor que dan los sensores si el brazo está parado, es decir, la velocidad angular es 0. Para corregirlo basta con estimarlo y restárselo a cada lectura.

Codigo 3.3: Eliminación del error de offset del giróscopo.

```
#include <Wire.h>
#include <L3G.h>

#define FACTOR_CONVERSION_245DPS 0.00875

#define DT 0.03               //Tiempo en segundos

int16_t dps_x;
int sampleNum = 500, offsetg_x = 0;;


void setup()
{

for(int n=0;n<sampleNum;n++){

```

```
17    gyro.read();
18    offsetg_x+=(int)gyro.g.x;
19  }
20  offsetg_x = offsetg_x/sampleNum;
21
22  }
23
24
25  void loop()
26  {
27    gyro.read();
28    dps_x =  (gyro.g.x-offsetg_y)*FACTOR_CONVERSION_245PS;
29    ...
30  }
```

Mediante el código 3.3 se obtiene la media del offset en sampleNum muestras y se consigue hacer despreciable el error de offset.

3.3.1.2 Deriva (Drift)

Para obtener la estimación del ángulo hay que realizar una integral, esto significa que cada iteración se suma un nuevo valor a la variable en la que se almacena el ángulo. Esta característica de la medida del ángulo provoca que si en cada medida hay un error pequeño se va a ir acumulando y tras varios segundos de ejecución de programa las medidas no van a ser válidas. En el caso del programa 3.2 se suma ese error cada 30 ms, lo que hace que tras 10 segundos de ejecución se haya sumado ese error 333 veces.

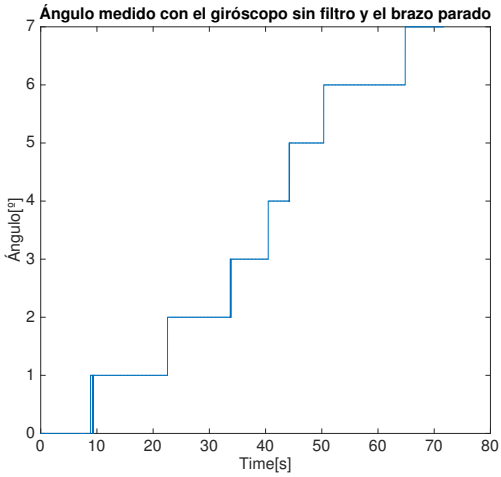
La única manera de reducir este error usando solo el giróscopo son los filtros paso-alto que incluye el modelo de giróscopo utilizado. Mediante este filtro se eliminan de la medida las muestras que tengan un valor muy pequeño como es el error por deriva. A pesar de utilizar este filtro no se consigue una medida válida de la posición del brazo; es necesario utilizar el acelerómetro para que la medida del ángulo sea precisa. La manera en la que se combinan los valores de los dos sensores se explicará en la sección 3.5.

3.3.1.3 Precisión

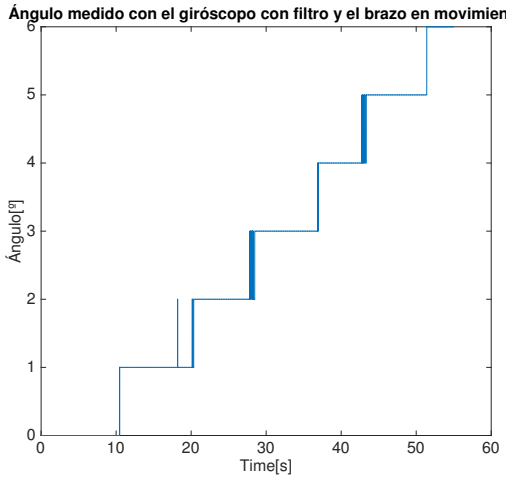
Además de los dos errores ya comentados hay otra característica del giróscopo, que no se puede considerar error pero sí limita la estimación del ángulo, y es que como este sensor mide la velocidad angular y ésta se integra la precisión de la medida es muy mala para movimientos lentos del brazo. Si la velocidad angular es muy baja el término que se va a sumar al ángulo es muy pequeño comparado con la suma total y no se va a percibir.

3.3.2 Medidas

En este apartado se van a mostrar ejemplos de medidas tomadas con los programas escritos en la sección 3.3. Las pruebas realizadas consisten en medir el ángulo con estos programas para poder observar las fuentes de error descritas en la sección 3.3.1. La primera prueba consiste en ejecutar el programa y dejar el brazo en reposo para comprobar si se produce error por deriva. La segunda prueba consiste en mover el brazo un ángulo conocido para comprobar la precisión de la medida.



(a) Angulo medido con el giróscopo sin filtro.



(b) Angulo medido con el giróscopo con filtro.

Figura 3.5: Resultados de la medida del ángulo con el giróscopo.

Se puede observar perfectamente el error de deriva, en menos de un minuto el error en la medida es de 7º, que es un error intolerable si se pretende estabilizar el brazo. En la gráfica del filtro software esos picos que se producen como el del segundo 20 son correcciones del filtro ante el pequeño incremento, pero la deriva que aparece sigue siendo muy elevada y no es una medida del ángulo aceptable.

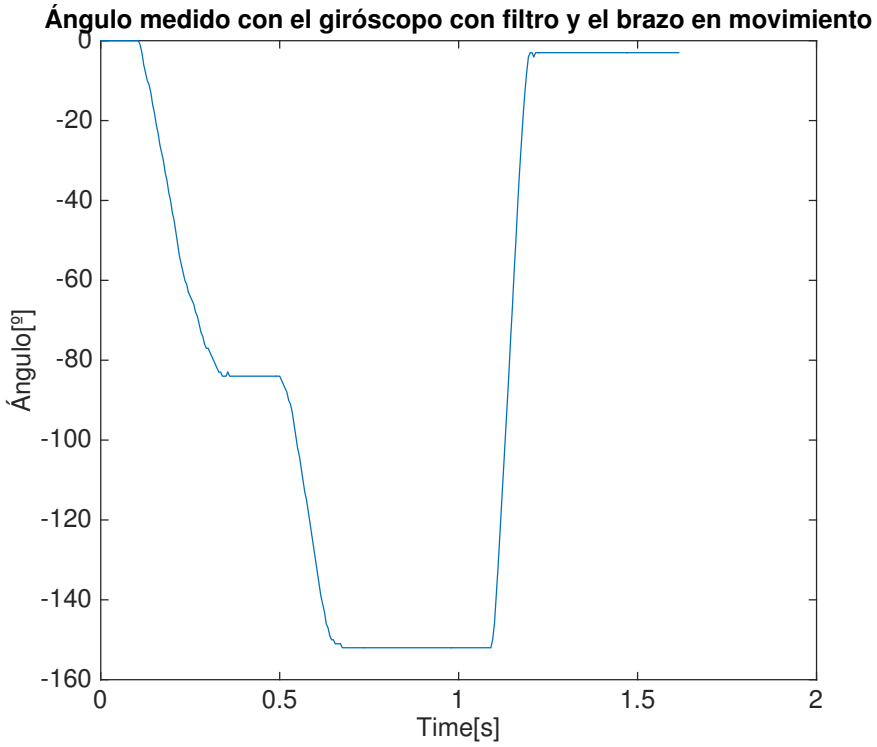


Figura 3.6: Ángulo medido con el giróscopo variando el ángulo.

La variación máxima real del ángulo es de 110º y después se ha vuelto a la posición inicial. El ángulo medido por el giróscopo es de 140º. Este error se debe a la baja precisión del giróscopo y a su principio de medida, que no le permite medir ángulos absolutos sino variaciones del ángulo.

3.4 Acelerómetro

Es cualquier instrumento destinado a medir las fuerzas de aceleración. Como se indica en [9, p. 2], hay muchas maneras diferentes de fabricar un acelerómetro. Algunos acelerómetros usan el efecto piezo-resistivo (contienen cristales microsópicos que se estiran debido a la aceleración y generan una tensión), otro tipo de tecnología es medir los cambios en la capacidad. Este último es el caso del acelerómetro utilizado , que utiliza la tecnología de fabricación MEMS[4]. Estos acelerómetros están compuestos de una masa móvil con unas placas que están unidas mediante un sistema de suspensión mecánica a un marco de referencia, como se muestra en la figura 3.7. Las placas móviles y las placas exteriores fijas forman condensadores. La desviación de la masa se mide usando la diferencia de capacidades.

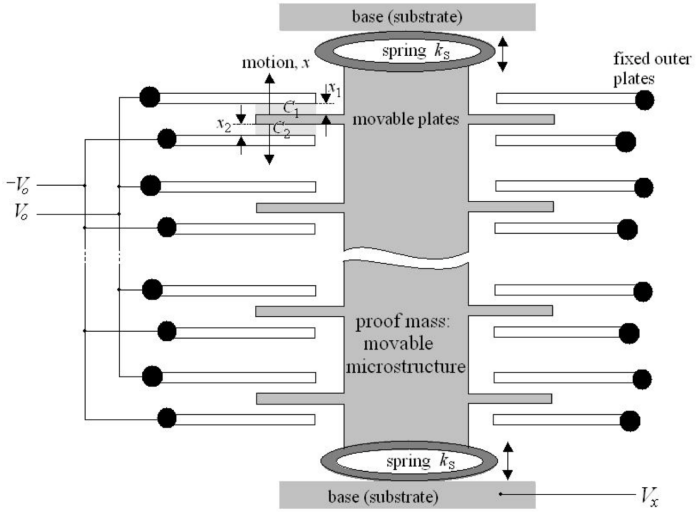


Figura 3.7: Acelerómetro de estructura vibrante.

Conociendo los tipos de acelerómetros y su funcionamiento ya se puede pasar a realizar el programa para tomar medidas con el modelo del acelerómetro utilizado en este trabajo. El modelo integrado en la IMU utilizada es el LSM303D de la marca STMicroelectronics, un acelerómetro de 3 ejes con salida digital. Se comunica mediante el bus de comunicación serie I^2C. Como ya se ha comentado previamente el fabricante del acelerómetro adquirido provee una librería para que la lectura de datos de los sensores de la IMU resulte más sencilla. A continuación se muestran los pasos a seguir para poder empezar a leer datos del sensor:

1. **Incluir la librería**: para poder utilizar las funciones y clases de datos definidas en la librería hay que incluir en el proyecto mediante un include.

2. **Declarar una variable**: en la librería hay definida una clase de datos llamada LSM303 que será la que usaremos para leer datos del acelerómetro, por lo tanto hay que declarar una variable de este tipo para llamar a las funciones de la librería.

3. **Inicializar**: el acelrómetro por defecto está configurado en modo "sleep"; hay que inicializarlo para poder empezar a leer datos. Además para poder comunicarse con Arduino hay que inicializar la comunicación serie.

4. **Lectura datos**: con los pasos previos ya realizados se pueden empezar a leer datos con la función *read* definida en la librería.

[4]Sistemas Microelectromecánicos

Un ejemplo de un programa mediante el que se podrían leer datos del acelrómetro se muestra en el código 3.4.

Codigo 3.4: Ejemplo de programa para tomar medidas del acelerómetro.

```
#include <Wire.h>               //Librería para la comunicación serie
#include <LSM303.h>                 //Librería del acelerómetro

LSM303 compass;                   //Declaración variable del tipo LSM303

void setup()
{
  Serial.begin(115200);             //Inicia el puerto serie
  Wire.begin();                  //Comunicación I2C entre IMU y Arduino

  compass.init();               //Inicialización del giróscopo
  compass.enableDefault();            //Habilita varios registros a su valor por defecto
  }


void loop()
{
  compass.read();
}
```

Estos valores leídos del sensor y almacenados en la variable *compass* son valores "raw", es decir, es un entero de 16 bits, no las fuerzas debidas a la aceleración. Para obtener la fuerza debida a la aceleración hay que realizar una serie de operaciones especificadas por el fabricante en la hoja de características del sensor. Para obtener valores de fuerza, medidos en g, se debe multiplicar el valor "raw" obtenido por un factor de conversión llamado sensibilidad de aceleración lineal, cuyas unidades son mg[5]/LSB. Los valores de estas constantes para cada FS se encuentran en [10, p. 10]. El procedimiento para obtener las fuerzas de aceleración es idéntico al que se ha llevado a cabo para obtener la velocidad angular con el giróscopo. Sin embargo, el objetivo final no es obtener las fuerzas de aceleración sino la posición, que es el ángulo. El ángulo se obtiene mediante los ángulos de Euler, que consiste en resolver unas matrices de rotación cuya solución son los ángulos de inclinación de un sólido rígido. La resolución de estas matrices se ha consultado en el documento del fabricante Freescale Semiconductor [11, p. 10]. La ecuación usada para estimar el ángulo es la ecuación 3.2, que en el documento equivale a la ecuación 37:

$$tan(angulo\_acelerometro\_x) = \frac{fuerza\_y}{\sqrt{fuerza\_x^2 + fuerza\_z^2}} \tag{3.2}$$

El resultado está en radianes, así que hay que pasarlo a grados con una constante que se llamará RAD_TO_DEG y cuyo valor es $\frac{180}{2\pi} = 57{,}2958$. Mediante este método sólo se obtienen ángulos entre -90º y +90º. Dado que el brazo sólo puede girar en un rango de entre -75º y +75º, el rango de medidas mediante este método es válido para este trabajo.

El programa para implementar esa ecuación es más sencillo que el programa necesario para estimar los ángulos mediante el giróscopo, pues en el caso del acelerómetro sólo hay que implementar una multiplicación. El código 3.5 implementa todos los pasos necesarios para estimar el ángulo mediante el acelerómetro.

[5]mg hace referencia a miligauss

Codigo 3.5: Estimación del ángulo mediante el acelerómetro.

```
#include <Wire.h>               //Librería para la comunicación serie
#include <LSM303.h>                 //Librería del acelerómetro

#define RAD_TO_DEG 57.2958
#define FACTOR_CONVERSION_2G 0.000061


LSM303 compass;                   //Declaración variable del tipo LSM303
double fuerza__x, fuerza_y, fuerza_z;
double angulo_acelerometro_x, raiz_x;
...


void loop()
{
    compass.read();

    fuerza_x =  -((compass.a.x)*FACTOR_CONVERSION_2G);//El signo - es debido a que la IMU está
         invertida
    fuerza_y =  -((compass.a.y)*FACTOR_CONVERSION_2G);
    fuerza_z =  -((compass.a.z)*FACTOR_CONVERSION_2G);

    raiz_x = sqrt((fuerza_x*fuerza_x)+(fuerza_z*fuerza_z));

    angulo_acelerometro_x = (atan2(fuerza_y,raiz_x))*RAD_TO_DEG;
}
```

Con el código 3.5 ya se obtiene una estimación del ángulo del brazo, sin embargo esa medida tiene fuentes de error que hacen que la precisión de la medida sea muy baja. Hay métodos para minimizar estos errores pero no se pueden eliminar completamente. A continuación se comentan estos errores y cómo reducirlos.

3.4.1 Fuentes de error

3.4.1.1 Offset

Este error se refiere al valor que dan los sensores si la fuerza en un eje es 0. Para corregirlo basta con estimarlo y restárselo a cada lectura.

Codigo 3.6: Eliminación del error de offset del acelerómetro.

```
#include <Wire.h>
#include <LSM303.h>

#define FACTOR_CONVERSION_2G 0.000061

double fuerza_x;
int sampleNum = 500, offset_x = 0;;


void setup()
{

for(int n=0;n<sampleNum;n++){

```

```
15    compass.read();
16    offsetg_x+=(int)compass.a.x;
17  }
18  offset_x = offset_y/sampleNum;
19
20  }
21
22
23  void loop()
24  {
25    compass.read();
26    fuerza_x =  -((compass.a.x-offseta_x)*FACTOR_CONVERSION_2G);
27    ...
28  }
```

Mediante el código 3.6 se obtiene la media del offset en sampleNum muestras y se consigue hacer despreciable el error de offset restando el valor de offset al valor de cada medida.

3.4.1.2 Ruido

En el caso del giróscopo se necesitaba movimiento del aparato en el que está acoplado el sensor para poder medir el ángulo, sin embargo con el acelerómetro se puede medir el ángulo en cualquier momento. El problema es que las medidas del acelerómetro introducen un ruido de baja frecuencia que provoca que no se pueda medir el ángulo con precisión en un corto período de tiempo. Las medidas del acelerómetro son útiles sólo para medir ángulos en un largo periodo de tiempo y de variaciones lentas. No hay manera de eliminar este error si sólo se usa para medir el acelerómetro. Para ello se combinan las medidas del acelerómetro y del giróscopo utilizando filtros.

3.4.2 Medidas

En este apartado se van a mostrar ejemplos de medidas tomadas con los programas escritos en la sección 3.4. Las prueba realizada consiste en medir el ángulo con estos programas para poder observar las fuentes de error descritas en la sección 3.4.1.

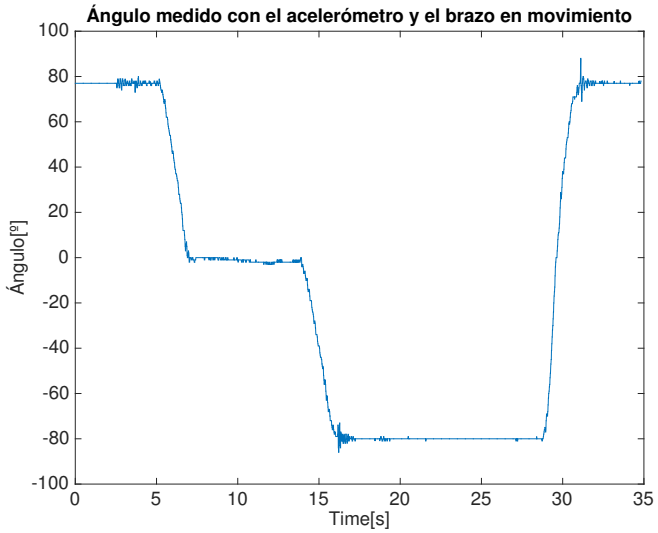


Figura 3.8: Resultados de la medida del ángulo con el acelerómetro.

La variación del ángulo en esta prueba es similar a la de la prueba realizada al medir con el giróscopo y moviendo el brazo. Como se ve en la gráfica3.8, al contrario de lo que sucedía con el giróscopo (cuyo punto de referencia estaba donde empezaba el brazo), los resultados ahora son de ángulos absolutos entre -90º y +90º, siendo 0º el ángulo en el que el brazo está paralelo al suelo. También se aprecia en la gráfica el error debido al ruido, cuando se produce un cambio brusco del ángulo de inclinación el ruido que aparece en las medidas es considerable. Ante cambios lentos del ángulo la precisión es elevada, pero es debido a las posibles variaciones rápidas del ángulo que se debe usar el giróscopo en conjunto. Para cuantificar mejor si ese ruido que aparece es importante se muestra en la figura 3.9 una ampliación de la anterior imagen.

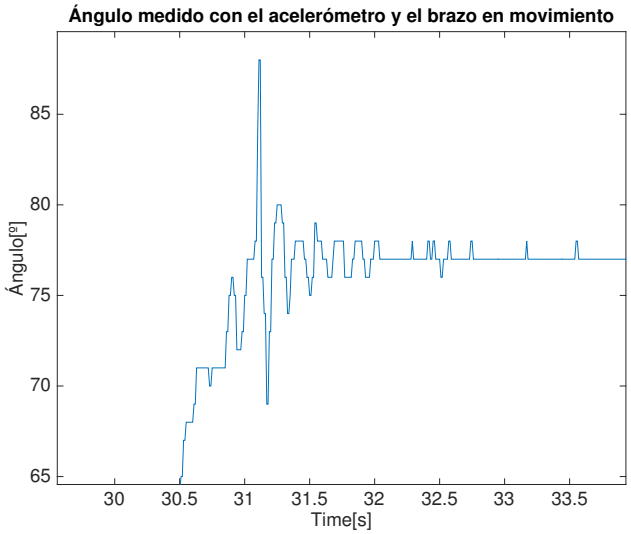


Figura 3.9: Ruido en la medida del ángulo con el acelerómetro.

Las variaciones del ángulo son de hasta 5º para un ángulo medido de 75º, si el ángulo fuese más pequeño, por ejemplo 10º, el error sería mucho mayor y correspondería a un 50 % de su valor. Con este error resultaría muy complejo estabilizar el brazo, es necesario el uso de filtros que combinen adecuadamente las medidas del acelerómetro y del giróscopo. Los distintos tipos de filtros se explicarán en la sección 3.5.

3.5 Filtros

Para que la medida del ángulo tenga una precisión suficiente para poder estabilizar el brazo es necesario combinar las medidas de los dos sensores explicados hasta ahora, el acelrómetro y el giróscopo. Esa es la utilidad de los filtros, mediante un algoritmo combina las medidas de ambos sensores para reducir o eliminar los errores típicos de cada sensor y dar una medida de alta precisión del ángulo. Los dos filtros más comunes y que mejor funcionan son el filtro de Kalman y el filtro complementario.

3.5.1 Filtro de Kalman

Este es el filtro más eficaz pero también el más complejo de implementar. Es un filtro recursivo que predice el error y lo corrige teniendo únicamente como datos las mediciones de entrada actuales, el estado calculado previamente y su matriz de incertidumbre. Para implementarlo se utiliza el control en el espacio de estados.

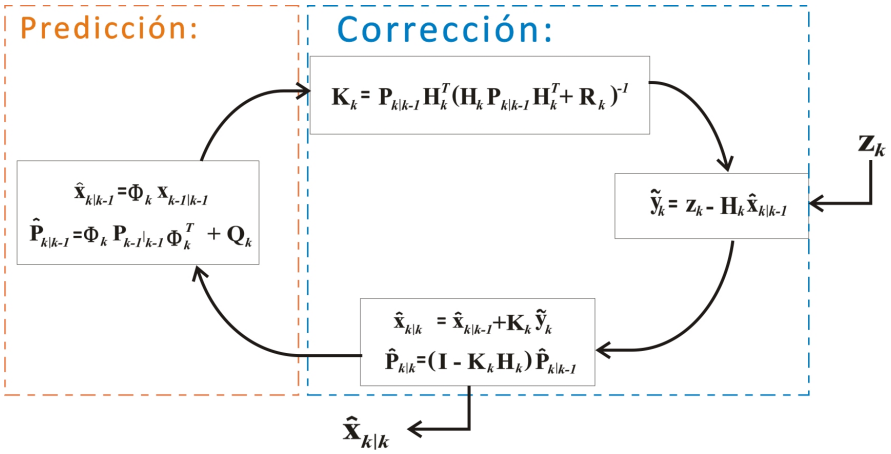


Figura 3.10: Algoritmo recursivo del filtro de Kalman.

Este filtro, debido a que trabaja con matrices, requiere mucha potencia de cálculo y el Arduino Uno no tiene un procesador muy potente.

3.5.2 Filtro complementario

El concepto del filtro complementario consiste en tomar las variaciones lentas del acelerómetro y las variaciones bruscas del giróscopo y combinarlas. El acelerómetro da buenas medidas de la posición en condiciones estáticas y el giróscopo las da en condiciones dinámicas, es decir, cuando el brazo se mueve rápido. La implementación de este filtro se realiza pasando las medidas del acelerómetro por un filtro paso-bajo y las medidas del giróscopo por un filtro paso-bajo y combinar el resultado para obtener la posición final. La implementación se realiza con una línea de código en el bucle principal, mostrada en el código 3.7.

Codigo 3.7: Línea de código para implementar el filtro complementario.

```
1 angulo_x = 0.95*(angulo_x + dps_x*DT) + 0.05*angulo_acelerometro_x;
```

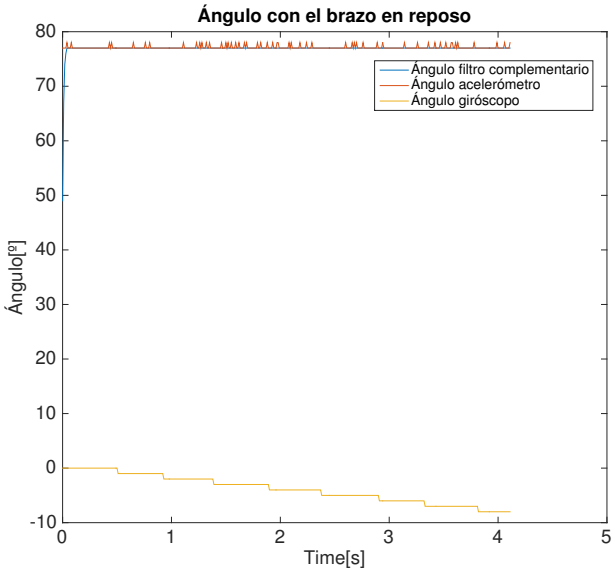
Teniendo los valores de los ángulos medidos con el giróscopo y el acelerómetro se introduce esa línea de código en el bucle principal y se obtiene una medida precisa del ángulo. Los valores de los filtros se pueden ajustar experimentalmente para obtener los mejores resultados.

Dado que la implementación de este filtro es muy simple y los resultados son satisfactorias para la obtención del ángulo del brazo se va a utilizar el **filtro complementario**.

3.5.3 Medidas

En este apartado se va a mostrar cómo los errores mostrados en los apartados del acelerómetro y del giróscopo se eliminan mediante el uso del filtro complementario. Los distintos experimentos realizados mostrarán la efectividad del filtro y la precisión en las medidas del ángulo del brazo.

El primer experimento consiste en dejar el brazo en reposo y registrar los ángulos del giróscopo y del acelerómetro y después obtener el del filtro complementario. El propósito de esta prueba es observar si el filtro corrige la deriva del giróscopo y si la medición del ángulo es correcta.



(a) Medidas del ángulo.

(b) Inclinación del brazo.

Figura 3.11: Resultados de la medida del ángulo con el giróscopo.

La figura 3.11 muestra cómo en la medida del filtro no hay deriva y además da una medida absoluta y precisa del ángulo. Cuando la variación del ángulo es pequeña el filtro paso-alto aplicado al giróscopo desprecia la medida del giróscopo , teniendo en cuenta únicamente la medida del acelerómetro.

El segundo experimento consiste en mover el brazo manualmente para comprobar los resultados de la medida del giróscopo, el acelerómetro y la combinación de ambos con el filtro complementario.

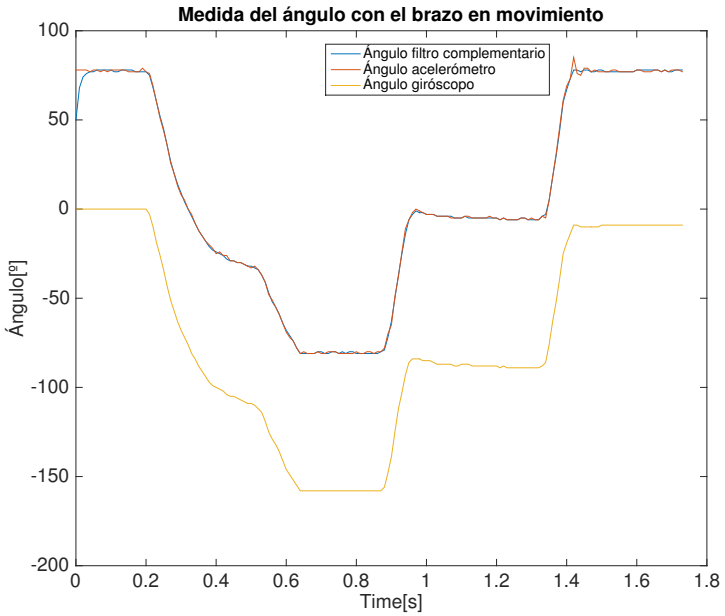


Figura 3.12: Medida del ángulo moviendo el brazo.

En la figura 3.12 se observa cómo el resultado obtenido con el filtro complementario sigue a las medidas dadas por el acelerómetro pero filtrando el ruido. Además también se comprueba que el giróscopo mide correctamente las variaciones del ángulo del brazo.

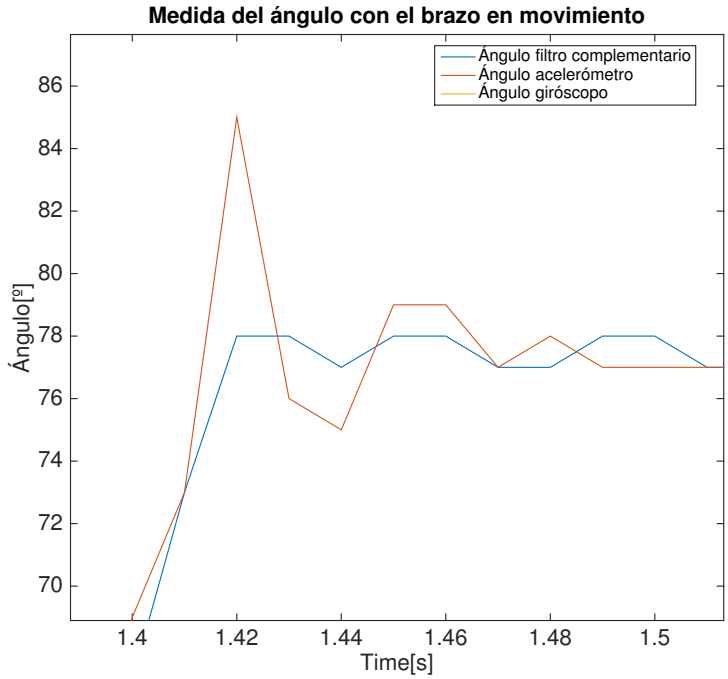


Figura 3.13: Ruido del acelerómetro.

Para observar el ruido se ha aumentado la imagen 3.12. Se comprueba cómo para un ángulo real del brazo de 78º el ruido que introduce el acelerómetro provoca que éste mida 86º, siendo un error elevado. Es en las variaciones bruscas del ángulo cuando la medida más fiable es la proporcionada por el giróscopo y se desprecian las medidas del acelerómetro ya que se ha aplicado un filtro paso-bajo.

En este apartado se ha mostrado el principio de funcionamiento de cada sensor y se han enumerado los errores que tiene cada uno al estimar el ángulo del brazo. Se ha mostrado experimentalmente la eficacia del filtro complementario, que corrige los errores típicos de cada filtro con una sola línea de código y requiriendo poco tiempo de cómputo, lo que hace que sea una buena opción utilizar este filtro en aplicaciones en las que no se disponga de un procesador muy potente.

CAPITULO IV

Motores y ESC

Capítulo 4

Motores y ESC

4.1 Introducción

Los motores acoplados a los extremos del brazo van a ser los actuadores mediante los que se va a estabilizar el ángulo. Ya que el brazo sólo puede moverse en dos direcciones de un solo eje se necesitan dos motores. Ya se ha explicado en el apartado 2.4 la razón de utilizar los controladores electrónicos de velocidad (ESC), en este apartado se explicará cuál debe ser la conexión entre los motores y el ESC y entre éstos y el Arduino.

4.2 Conexión de los motores y los ESC

El esquema de la conexión, realizado con Fritzing [12], entre batería, ESC, Arduino y los motores se muestran en la figura 4.1.

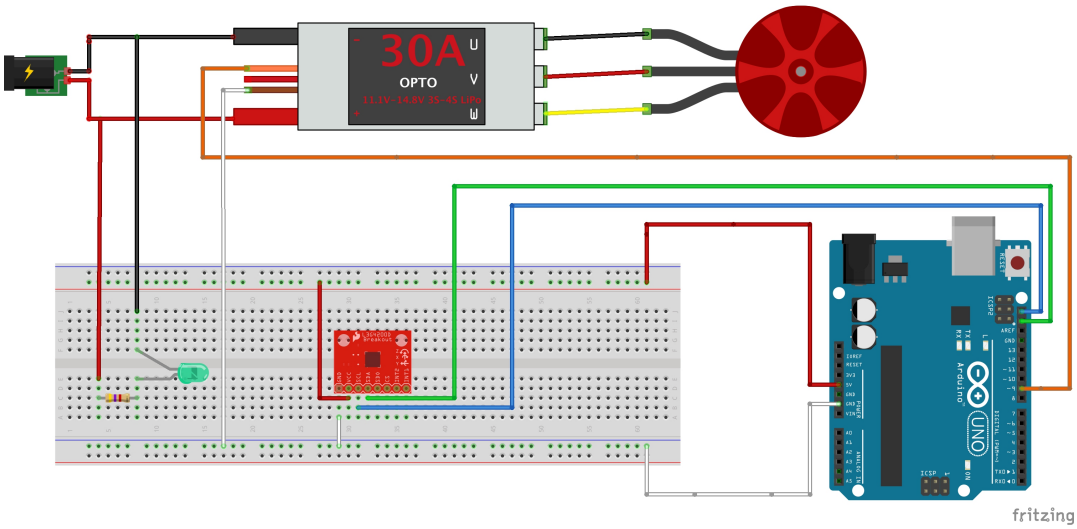


Figura 4.1: Esquema de conexión motores.

Sólo se muestra un motor porque la conexión del otro es idéntica. Como los motores giran a muchas revoluciones y las hélices tienen una sección muy fina son peligrosas cuando están en movimiento. Para asegurarse de que no les está llegando corriente a los motores y no van a empezar a girar se ha conectado un LED verde que se encenderá cuando se conecte el interruptor de la batería. La señal de control del

ESC debe llevarse a una salida digital y la masa de los pines de control del ESC debe conectarse al pin GND del Arduino. **Nunca se debe conectar el pin rojo de la señal de control al pin de 5 V del Arduino**. Como se indica en [13] no hay protección frente en ese pin del Arduino. Este pin está conectado directamente al microcontrolador ATmega328, la interfaz de USB del microcontrolador y al regulador de 5 V, siendo la tensión máxima soportada por todos ellos 6 V.

Antes de conectar los motores hay que asegurarse de que ambos giran en el mismo sentido. Si cada motor gira en un sentido ambos motores van a empujar al brazo en la misma dirección, siendo imposible estabilizarlo una vez pasado el punto de equilibrio. Si por ejemplo el motor izquierdo empuja el brazo hacia arriba y el motor derecho empuja el brazo hacia abajo sólo se puede actuar sobre el brazo girándolo hacia la derecha. Si la conexión realizada hace girar a cada motor en un sentido sólo se debe **intercambiar la conexión de dos fases** en un motor y el problema está solucionado.

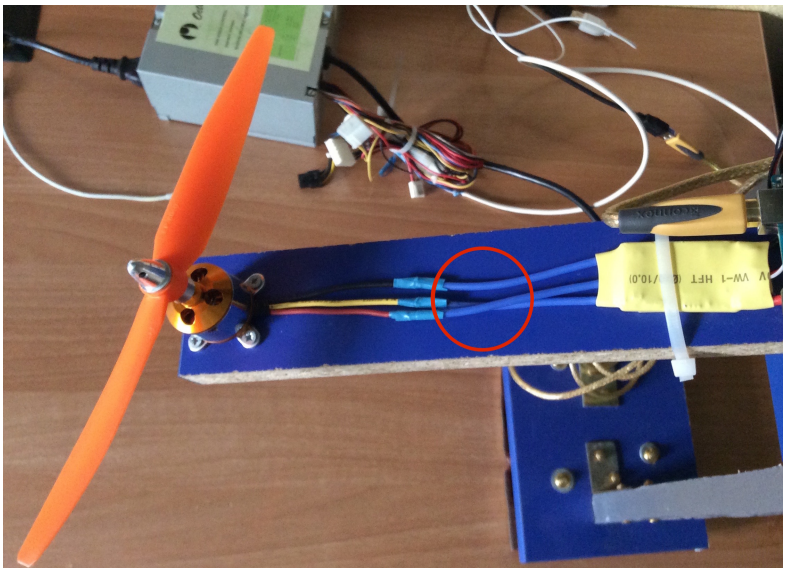


Figura 4.2: Fases intercambiadas en uno de los motores.

Con las conexiones de los motores ya realizadas se pueden empezar a realizar pruebas con los motores para conocer su funcionamiento con el objetivo de que los reguladores que se implementen tengan un funcionamiento óptimo.

4.3 Regulación de velocidad de los motores

La señal mediante la que se regulan los motores es una señal PPM (pulse position modulation), como se ha explicado en la sección 2.4. Este apartado se va a centrar en explicar cómo se realiza el programa necesario para generar esas señales y enviarlas al ESC. Tiene varias modos de funcionamiento, entre los que se incluye el modo acelerador o *throttle* y el modo freno. Para que los motores se muevan se debe programar el ESC en modo *throttle*. Una peculiaridad de estos controladores es que la elección del modo de funcionamiento se realiza mediante sonidos, justo después de escuchar el sonido asociado al modo de funcionamiento que se quiere elegir se debe enviar la señal mínima al controlador y el ESC quedará programado en ese modo. El modo *throttle* corresponde al segundo sonido emitido por el ESC, el primero indica que está conectado. Para enviar señales al ESC se debe hacer como si fuese un servo, utilizando la librería servo de Arduino. Los pasos necesarios para llegar a arrancar un motor son los siguientes:

1. **Incluir la librería**: para poder utilizar las funciones y clases de datos definidas en la librería servo hay que incluirla en el proyecto.
2. **Declarar una variable**: en la librería servo hay una clase de datos llamada *servo*, hay que declarar una variable por cada motor.
3. **"Adjuntar" el motor**: mediante la función *attach* declarada en la librería servo se especifica en qué pin está conectado cada motor.
4. **Programación del ESC**: para iniciar el modo programación del ESC hay que escribir la señal máxima de los motores, que son 2 ms.
5. **Elección modo *throttle***: el modo de funcionamiento se elige mediante sonidos, cuando se oiga el sonido asociado al modo en el que se quiere trabajar se quiere enviar la señal mínima, que son 0.7 ms.
6. **Escribir un valor de velocidad en los motores**: el último paso es escribir el valor de velocidad a la que se quiere que giren los motores, ese valor deberá estar comprendido entre 1 y 2 ms.

Un ejemplo de un programa mediante el que se arrancan los motores se muestra en el código 4.1. En él se han programado todos los pasos arriba mencionados para poder la velocidad de los motores.

Codigo 4.1: Ejemplo de programa para arrancar los motores.

```
#include <Servo.h>                //Librería para para controlar un servo


#define MAX_SIGNAL                 2000
#define MIN_SIGNAL                 700

#define MOTOR_IZQUIERDO_PIN        9

Servo motorizquierdo;       //Definición de la variable para poder adjuntar el motor
int velocidad_motor_izquierdo;

void setup()
{
  //Adjuntar el motor, se asocia la variable motorizquierdo con el motor conectado al pin 9
    motorizquierdo.attach(MOTOR_IZQUIERDO_PIN);

    //Ahora se debe entrar en modo programación, para lo que se escribe la señal máxima en el
        ESC
      motorizquierdo.writeMicroseconds(MAX_SIGNAL);

    // Esperando a que se pulse una tecla
        while (!Serial.available());
        Serial.read();

    // Enviar valor minimo
        motorizquierdo.writeMicroseconds(MIN_SIGNAL)
  }

void loop()
{
  velocidad_motor_izquierdo = 1300;
  motorizquierdo.writeMicroseconds(velocidad_motor_izquierdo);
}
```

El bucle while de la línea 18 está ejecutándose hasta que se pulse una tecla cualquiera. Se coloca para que cuando se escuche el sonido correspondiente al modo de funcionamiento del controlador con el que se quiere trabajar se pulse cualquier tecla y el programa siga ejecutándose. Para mandar un valor de velocidad al controlador se utiliza la función *writeMicroseconds*. Como el valor pasado son microsegundos hay que multiplicar por 1000 el valor de milisegundos correspondiente.

CAPITULO V

Estabilización del brazo

Capítulo 5

Estabilización del brazo

5.1 Introducción

Hasta este punto el trabajo se ha centrado en estudiar los componentes y cómo utilizarlos para tomar medidas con los sensores y variar la velocidad de los motores, pero aún no se ha aplicado la teoría de control sobre ningún componente. En este apartado se va a aplicar la teoría de control implementando distintos controladores para estabilizar el brazo en un ángulo que será indicado al sistema como referencia.

5.2 Controladores

Un controlador tiene la función de mantener constante una característica determinada del sistema. En este caso se mantendrá constante el ángulo del brazo midiendo la posición mediante la IMU y actuando mediante los motores cuando haya error. Existen muchos controladores pero los más comunes son los controladores PID. en este trabajo se implementarán, además del controlador PID, otros controladores para comprobar cómo mejoran o empeoran la respuesta del sistema en lazo cerrado respecto al controlador PID. Se probarán los controladores PPI, PIP, PIDPID. Para cada controlador se realizarán pruebas experimentales variando los valores de las constantes y midiendo parámetros que determinan la calidad de la respuesta como el tiempo de establecimiento, el tiempo de subida... En adelante se explicará cómo implementarlos y se mostrarán los resultados de cada uno.

5.2.1 PID

Es un mecanismo de control sobre un bucle de control por realimentación. Consiste en una combinación de 3 controladores independientes. Estos 3 controladores en conjunto procesan el error calculado a partir de la señal de referencia menos la señal de salida real. Con cada uno de los controladores se mejora uno de los dos regímenes de un sistema de control, bien la respuesta en régimen transitorio o bien la respuesta en régimen permanente, mejorando la estabilidad, la precisión y la velocidad de respuesta. Como se indica en [14, p. 669], el algoritmo del control PID consiste en la suma de los tres controladores mencionados, su esquema se muestra en la figura 5.1.

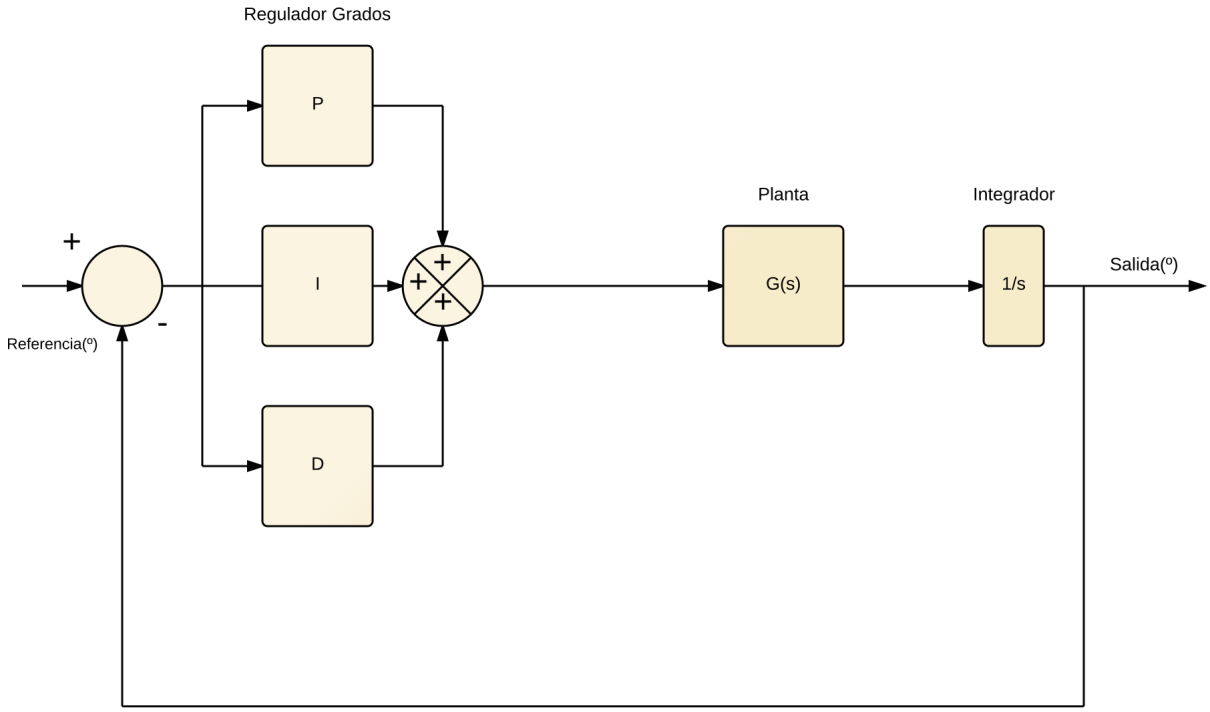


Figura 5.1: Esquema del sistema.

Cada controlador puede expresarse de distinta manera en función de si se trabaja en el dominio de Laplace o en el dominio del tiempo, aplicando las propiedades de la transformada de Laplace se pasará del dominio del tiempo al dominio de Laplace para mostrar ambas expresiones:

- **Proporcional**: su respuesta es proporcional al error. Si sólo se utiliza este controlador hay mucho ruido en el sistema y el sistema oscilaría mucho hasta estabilizarse. A mayor valor de la constante K_p mejor es la precisión y peor es la estabilidad, es decir, mejor es la respuesta en régimen permanente y peor es la respuesta en régimen transitorio.

$$\frac{u(t)}{e(t)} = K_p \tag{5.1}$$

$$\frac{U(s)}{E(s)} = K_p \tag{5.2}$$

- **Integral**: su respuesta es proporcional a la integral del error, es decir, a la variación del error. Se puede considerar una memoria que estima lo que va a pasar en función de los errores pasados. Elimina el error en régimen estacionario a costa de aumentar el tiempo de establecimiento y hacer la respuesta más lenta.

$$u(t) = K_i \int_0^t e(\tau)\, d\tau \tag{5.3}$$

Por la propiedad de la integral de la tabla de propiedades de la transformada de Laplace se sabe que una integral en el dominio del tiempo equivale a una división por s en el dominio de Laplace, por lo tanto:

$$\frac{U(s)}{E(s)} = \frac{K_i}{s} = K_p \frac{1}{T_i s} \tag{5.4}$$

- **Derivativo**: su respuesta es proporcional a la derivada del error, es decir, a la velocidad de variación del error. El efecto derivativo mejora la respuesta transitoria del sistema pero empeora la respuesta en régimen permanente. En particular aumenta la velocidad de respuesta del sistema pero introduce ruido a alta frecuencia, que se disminuye mediante la acción del proporcional.

$$u(t) = K_d \frac{de(t)}{dt} \tag{5.5}$$

Por la propiedad de la derivada de la tabla de propiedades de la transformada de Laplace se sabe que una derivada en el dominio del tiempo equivale a una multiplicación por s en el dominio de Laplace, por lo tanto:

$$\frac{U(s)}{E(s)} = K_d s = K_p T_d s \tag{5.6}$$

Teniendo la función de transferencia de cada controlador que compone el PID se puede obtener su función de transferencia sumando cada componente:

$$PID = \frac{U(s)}{E(s)} = P + I + D = K_p + \frac{K_i}{s} + K_d s = K_p(1 + \frac{1}{T_i s} + T_d s) \tag{5.7}$$

Estas dos formas de representar la expresión de un PID se denominan paralela (la expresión que tiene las constantes K) y estándar (la expresión que incluye las constantes T_i y T_d).

La entrada de referencia es el ángulo en grados al que se quiere estabilizar el brazo. Para estabilizar el brazo se va a establecer una velocidad base de 1,3 ms para cada motor, que será la velocidad a la que giren los motores si el error es nulo. Si hay error el PID actuará acelerando un motor y desacelerando el otro para corregirlo. Los pasos a seguir para implementar este controlador son:

1. **Definir las constantes**: hay que definir los valores de las constantes que se multiplicarán a cada parámetro del controlador, después se ajustarán experimentalmente.

2. **Calcular el error**: como ya se ha calculado el ángulo del brazo en el capítulo 3 ahora se necesita un dato de entrada, el ángulo de referencia. Para empezar se escribirá en en el programa como un *# define*, más adelante se modificará desde LabView. Para calcular el error sólo hay que restar el ángulo de referencia al ángulo medido del brazo.

3. **Calcular el integral**: como el parámetro es una integral es imprescindible multiplicar por DT[1] la suma del error.

4. **Calcular el derivativo**: ya que es una derivada hay que dividir por DT la resta del error actual y el error de la iteración anterior.

5. **Cálculo del PID**: teniendo los 3 parámetros hay que combinarlos todos en una única variable.

6. **Aplicar el control a los motores**: como se indica en la sección 4.2 cada motor gira en un sentido, por lo tanto a un motor hay que sumar el valor del controlador y al otro restarlo.

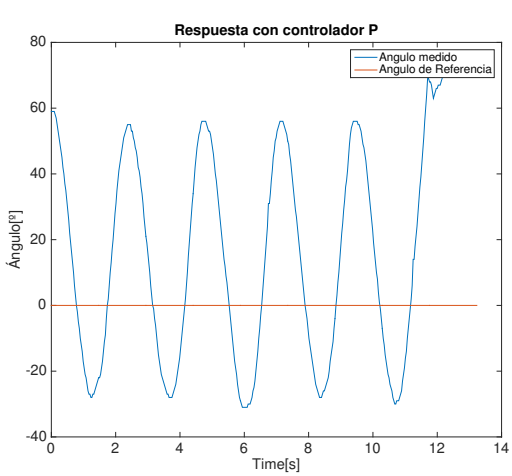
[1] Delta Time: es la duración del bucle principal en segundos

Codigo 5.1: Implementación del PID.

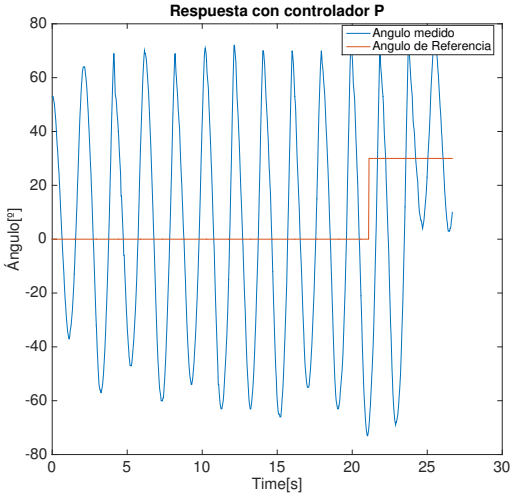
```
#include <Servo.h>
#include <Wire.h>
#include <LSM303.h>
#include <L3G.h>

#define DT 0.03
#define ANGULO_REFERENCIA = 0; //Ángulo al que se quiere estabilizar el brazo

#define MAX_SIGNAL                2000
#define MIN_SIGNAL                700
#define MAX_VEL                   1500
#define MIN_VEL                   1000
#define MOTOR_IZQUIERDO_PIN       9
#define MOTOR_DERECHO_PIN         10
#define Throttle_Motor_Izquierdo  1300   //Velocidad de los motores si el error es 0
#define Throttle_Motor_Derecho    1300

double angulo_x;

//VARIABLES MOTORES

  Servo motorizquierdo, motorderecho;
  int velocidad_motor_izquierdo, velocidad_motor_derecho;

//VARIABLES PID

    float Kp = 6, Ki = 3, Kd = 2;
    float error, error_pasado, derivativo, integral, PID_angulo;

void setup()
{
    Serial.begin(115200);
        Wire.begin();
  //Adjuntar motores, programar ESC's y habilitar sensores
}
void loop()
{
  //Medida ángulo
    ...
   angulo_x = 0.95*(angulo_x + dps_x*DT) + 0.05*angulo_acelerometro_x;

  //PID

   error = ANGULO_REFERENCIA - angulo_x;
       integral+=(error*DT);
       derivativo = (error-error_pasado)/DT;
       PID_angulo = Kp*error + Ki*integral + Kd*derivativo;

  //Control motores

   velocidad_motor_izquierdo = Throttle_Motor_Izquierdo + PID_angulo;
     velocidad_motor_derecho   = Throttle_Motor_Derecho - PID_angulo;
     motorizquierdo.writeMicroseconds(velocidad_motor_izquierdo);
     motorderecho.writeMicroseconds(velocidad_motor_derecho);

     error_pasado = error;  //Para calcular el derivativo
}
```

El código 5.1 muestra cómo se implementa el controlador PID. Las constantes MAX_VEL y MIN_VEL se utilizan para limitar la velocidad de los motores desde la línea 64 del código hasta la línea 69. Es recomendable para que la velocidad no sobrepase el límite, si desciende de 1 ms el motor se puede parar y si aumenta de 2 ms el motor puede llegar a romperse. Se ha establecido en 1,5 ms la máxima porque la fuente de alimentación no suministra suficiente corriente para llevar a un motor a la velocidad máxima mientras el otro sigue girando.

A continuación se va a mostrar el resultado de la estabilización con las distintas partes que combinan este controlador. Primero se utilizará sólo el controlador proporcional (P), después el controlador proporcional-integral (PI) y, por último, se implementará el controlador proporcional-integral-derivativo (PID).



(a) Respuesta prueba 1.



(b) Respuesta prueba 2.

Figura 5.2: Respuesta del sistema con controlador P.

Los valores de las constantes para las gráficas de la figura 5.2 son $\mathbf{K_p = 3}$ en el **caso (a)** y $\mathbf{K_p = 4}$ en el **caso (b)**. En ambos casos la respuesta del sistema es oscilante, al no disponer de un controlador que recuerde los errores anteriores y otro que varíe en función de la velocidad de variación del error el sistema es incapaz de llegar a estabilizarse, una vez que actúa el controlador se pasa del ángulo de referencia en ambos sentidos del brazo.

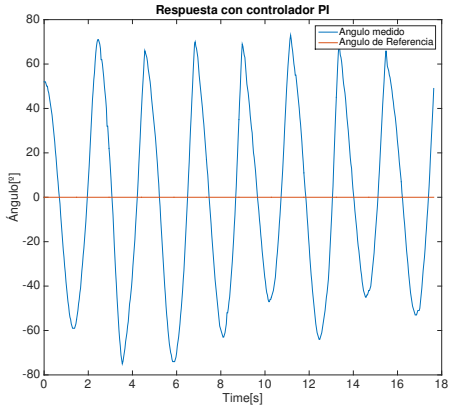
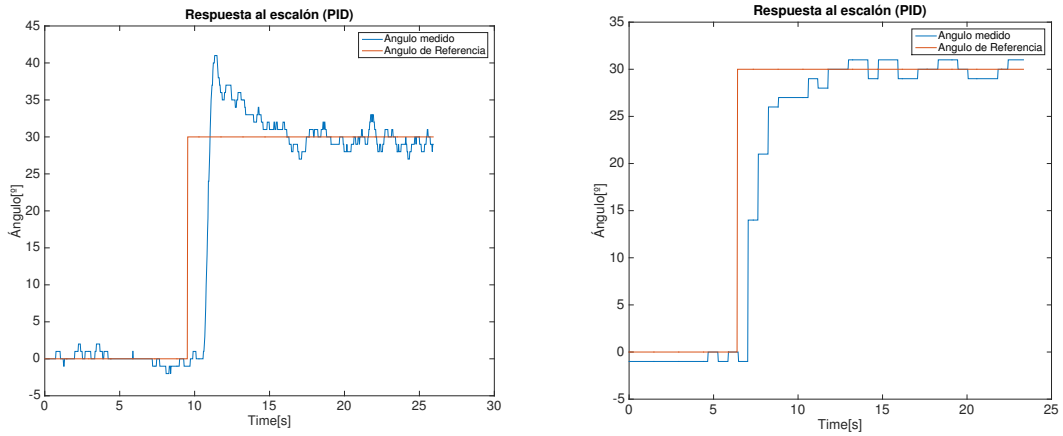


Figura 5.3: Respuesta del sistema para controlador PI.

Los valores de las constantes para la gráfica de la figura 5.3 son $\mathbf{K_p = 3}$ y $\mathbf{K_i = 1}$. Con este controlador tampoco se consigue estabilizar el brazo ya que no se dispone de ninguna medida de la velocidad de variación del error como la obtenida con el controlador derivativo.

Para comprobar cómo varía la respuesta simplemente variando los valores de las constantes se van a realizar dos pruebas variando las tres constantes del controlador. El resultado de la estabilización con este controlador para las dos pruebas se muestra en la figura 5.4.

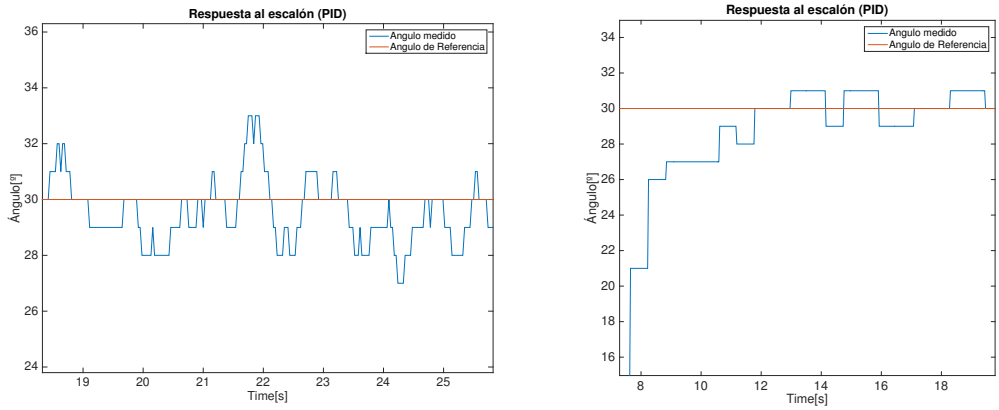


(a) Respuesta en la prueba 1.

(b) Respuesta en la prueba 2.

Figura 5.4: Respuesta del sistema para controlador PID.

Los valores de las constantes para las gráficas de la figura 5.4 son $\mathbf{K_p = 6}$, $\mathbf{K_i = 3}$ y $\mathbf{K_d = 2}$ para el **caso (a)** y $\mathbf{K_p = 5}$, $\mathbf{K_i = 0{,}5}$ y $\mathbf{K_d = 4}$ para el **caso (b)**. El ruido presente en ambas respuestas tiene dos orígenes, el primero son las vibraciones provocadas por el giro de los motores, cuando el brazo llega al ángulo de referencia la vibración de los motores provoca un pequeño error que el controlador intenta corregir constantemente, y el segundo a la constante K_d , aumentándola se consigue un mayor tiempo de respuesta pero también se introduce más ruido en el sistema. Sin embargo la magnitud de ruido introducida por los motores es mucho mayor que la que introduce el aumento de la constante ya que este último se puede contrarrestar (como se observa en la el caso (b)) mediante un buen ajuste de las constantes. En la figura 5.5 se muestra la magnitud del ruido después del escalón.



(a) Error en la prueba 1.

(b) Error en la prueba 2.

Figura 5.5: Detalle del ruido presente en la respuesta en régimen permanente del controlador PID.

Para poder comparar los resultados con el resto de controladores se van a calcular en la respuesta de

cada controlador los parámetros en régimen transitorio. La manera de calcular el valor exacto de algunos de ellos es mediante un script de Matlab. El código 5.2 permite obtener el tiempo de establecimiento, el tiempo de pico y el sobreimpulso máximo con exactitud.

Codigo 5.2: Cálculo de parámetros en régimen transitorio con Matlab.

```
figure(1);
plot(dt, angulomedido, dt, anguloreferencia);
title('Respuesta al escalón');
xlabel('Time[s]');
ylabel('Ángulo[°]');
legend('Ángulo medido', 'Ángulo de referencia');

m = 1031;
while angulomedido(m) > 27 & angulomedido(m) < 33
 m = m-1;
end
Tiempo_de_establecimiento = dt(m)-9.5;
[Mp, k] = max(angulomedido);
Tiempo_de_pico = dt(k)-9.5;
```

El valor de la variable m tiene que ser el número de muestras de *angulomedido*, se recorre el array *angulomedido* desde la última muestra y cuando su valor deja de estar en el 10 % del ángulo de referencia. La constante 9.5 restada a los tiempos de establecimiento y de pico es el instante en que comienza el escalón, que es cuando hay que medir los parámetros. El tiempo de retardo y el tiempo de subida se puden obtener con el puntero de Matlab, situándose sobre el punto adecuado, el momento en que se llega al 50 % del valor del valor máximo en el caso del tiempo de retardo y el momento en el que se llega por primera vez al valor de escalón en elk caso del tiempo de subida. En el caso del controlador PID los valores de estos parámetros son:

Tabla 5.1: Parámetros en régimen transitorio PID.

Parámetro	Caso (a)	Caso (b)
Tiempo de retardo (t_d)	2 seg	1.2 seg
Tiempo de subida (t_r)	3 seg	5.4 seg
Tiempo de pico (t_p)	1.84	6.59 seg
Tiempo de establecimiento ($t_{s10\,\%}$)	14.83 seg	4.19 seg
Máximo sobreimpulso (M_p)	41º (1.36)	31º (1.03)

5.2.2 PPI

En este controlador se incluye un nuevo bucle de realimentación. Ahora se realimenta también la velocidad angular, que es la salida de la planta, para aplicar un regulador. El esquema de este regulador se muestra en la figura 5.6 y su implementación en el código 5.3.

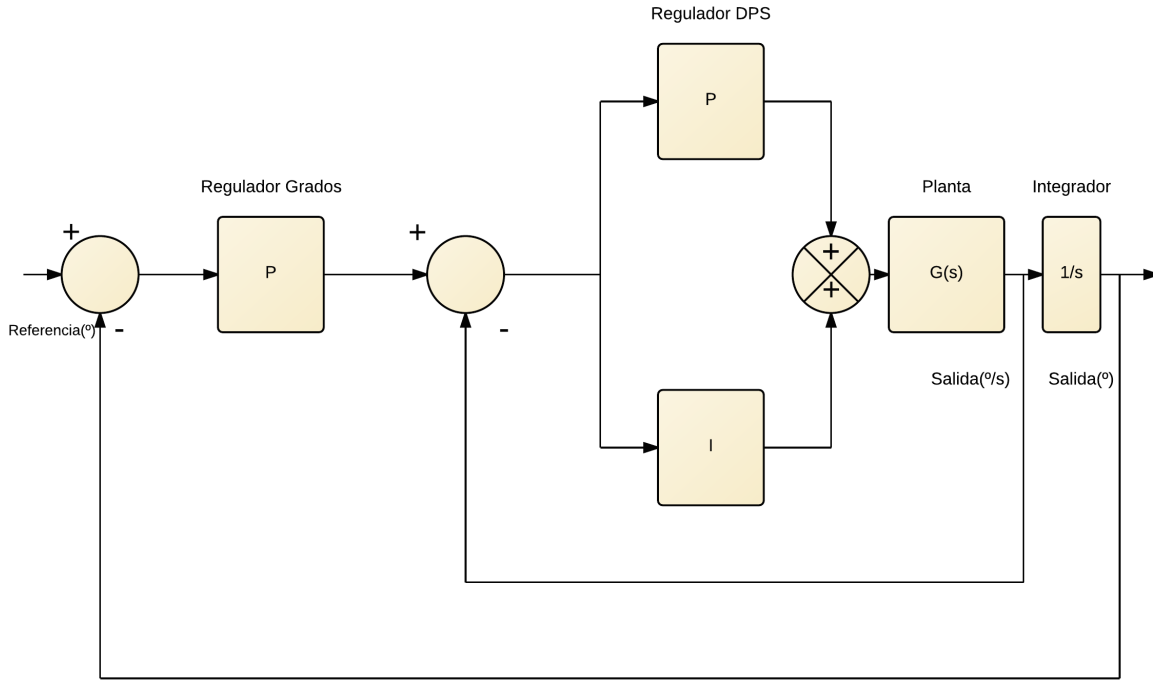
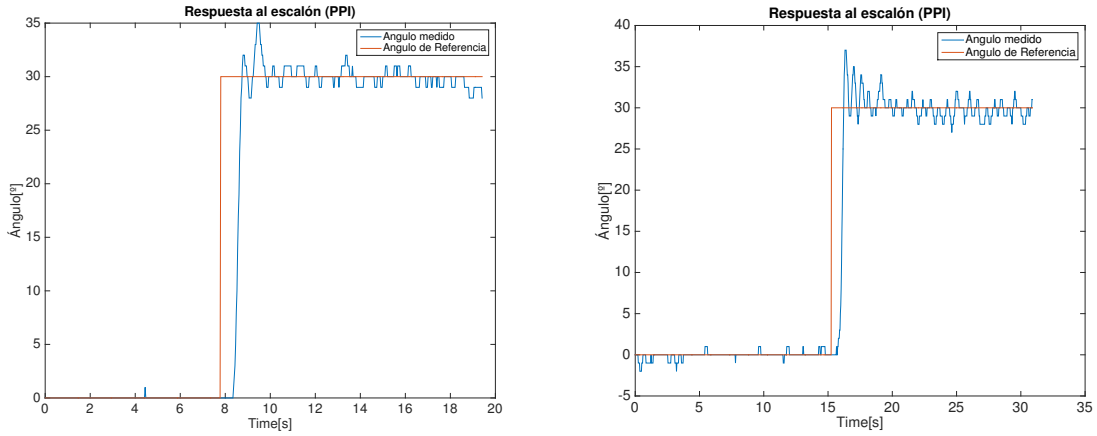


Figura 5.6: Esquema del controlador PPI.

Codigo 5.3: Implementación del PPI.

```
//Igual que en el PID

//VARIABLES PPI

    float Kp = 6, Kpi_p = 3, Kpi_i = 2;
    float error_p, error_pi, integral, PPI_angulo;

void setup(){
    Serial.begin(115200);
        Wire.begin();
  //Adjuntar motores, programar ESC's y habilitar sensores
}
void loop(){
  //Medida ángulo
    ...
   angulo_x = 0.95*(angulo_x + dps_x*DT) + 0.05*angulo_acelerometro_x;

  //PPI

    error_p = ANGULO_REFERENCIA - angulo_x;
    error_pi = Kp*error_p - dps_x;
    integral+=(error_pi*DT);
    PPI_angulo = Kpi_p*error_pi + Kpi_i*integral;

  //Control motores

   velocidad_motor_izquierdo = Throttle_Motor_Izquierdo + PPI_angulo;
     velocidad_motor_derecho   = Throttle_Motor_Derecho - PPI_angulo;

    //El resto igual que en el PID
}
```

A partir de la línea 50 está la implementación del controlador, primero se calcula el error de la salida en grados y después a ese error multiplicado por la constante K_p se le resta la salida en grados por segundo. El resultado de la estabilización con este controlador para distintos valores de las constantes se muestra en la figura 5.7.

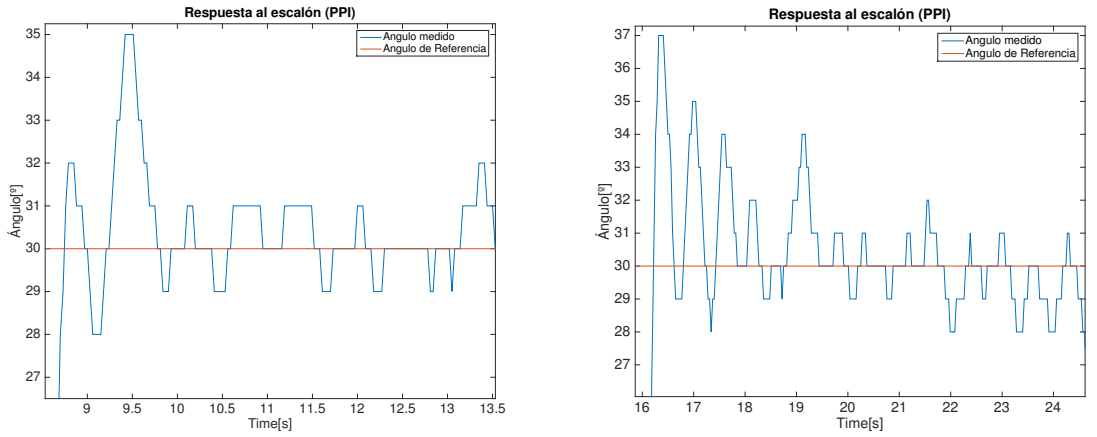


(a) Respuesta en la prueba 1.

(b) Respuesta en la prueba 2.

Figura 5.7: Respuesta del sistema para controlador PPI.

Los valores de las constantes para las gráficas de la figura 5.7 son $\mathbf{K_p = 3}$, $\mathbf{K_{pi\_p} = 4}$ y $\mathbf{K_{pi\_i} = 2}$ para el **caso (a)** y $\mathbf{K_p = 4}$, $\mathbf{K_{pi\_p} = 5}$ y $\mathbf{K_{pi\_i} = 2}$ para el **caso (b)**. A simple vista con este controlador se ha mejorado la velocidad de respuesta con respecto al PID y el ruido en régimen permanente es menor. Además en el caso (a) el error en régimen permanente antes del escalón es prácticamente nulo, siendo el error máximo de 1º en un solo instante. El ruido en régimen permanente se muestra en detalle en la figura 5.8.



(a) Error en régimen permanente en la prueba 1.

(b) Error en régimen permanente en la prueba 2.

Figura 5.8: Detalle del ruido presente en la respuesta en régimen permanente del controlador PPI.

Con el ajuste de las constantes se ha conseguido mejorar la velocidad de respuesta, llegando antes en el caso (a) al régimen permanente y con un máximo sobreimpulso menor. Los parámetros en régimen transitorio de este controlador son:

Tabla 5.2: Parámetros en régimen transitorio PPI.

Parámetro	Caso (a)	Caso (b)
Tiempo de retardo (t_d)	0.83 seg	1.1 seg
Tiempo de subida (t_r)	0.97 seg	1.2 seg
Tiempo de pico (t_p)	1.65 seg	1.32 seg
Tiempo de establecimiento ($t_{s10\,\%}$)	1.83 seg	9.63 seg
Máximo sobreimpulso (M_p)	35º (1.16)	37º (1.23)

5.2.3 PIP

Este controlador es muy similar al implementado en el apartado anterior, pero ahora se realiza un controlador proporcional integral para los grados y un controlador proporcional para la velocidad angular. El esquema de este regulador se muestra en la figura 5.9

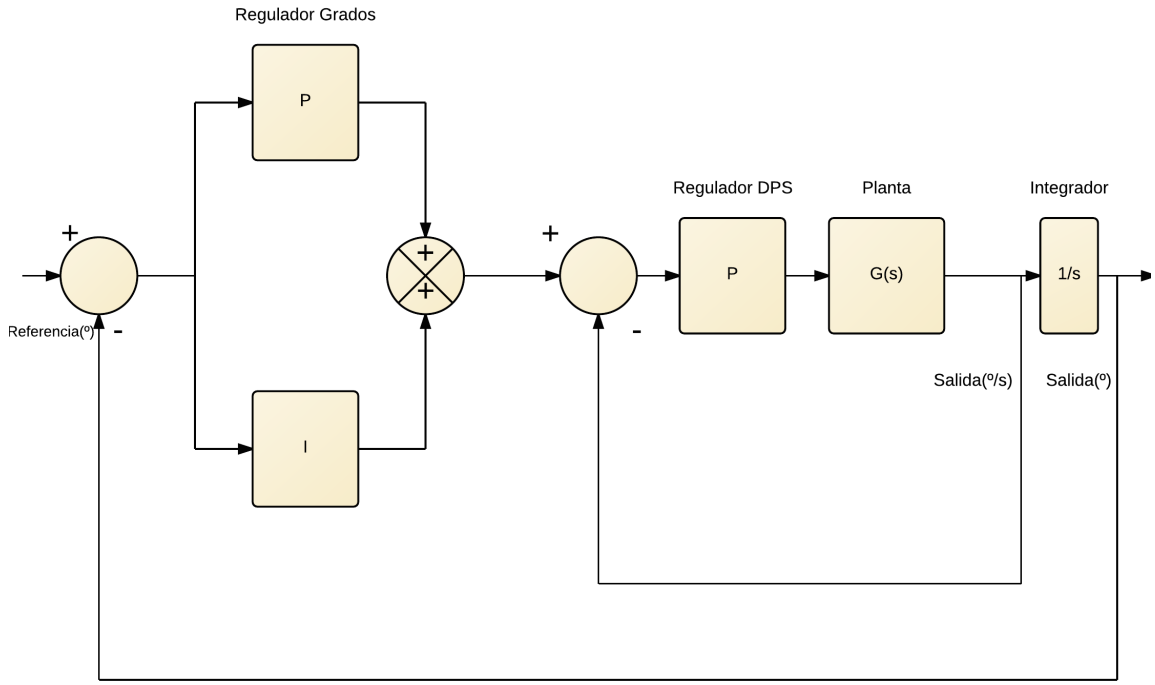


Figura 5.9: Esquema del controlador PIP.

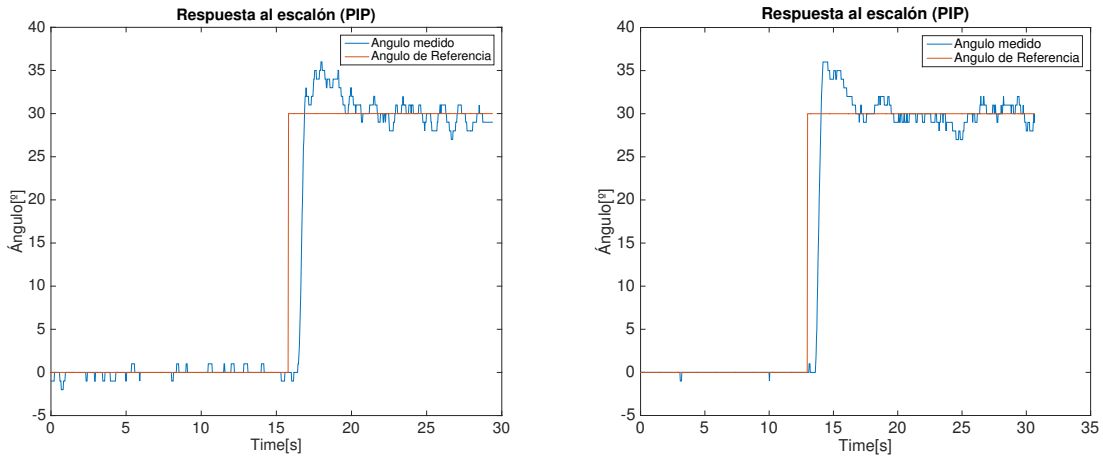
Este controlador consiste en realimentar la salida en grados aplicándole al error un controlador proporcional y realimentar la salida de la planta en grados por segundo y al error aplicarle un controlador proporcional integral. La implementación de este controlador se realiza mediante el código 5.4.

Codigo 5.4: Implementación del PIP.

```
1  //Igual que en el PID
2
3  //VARIABLES PIP
4
5      float Kpi_p = 3, Kpi_i = 1,  Kp_p = 4;
6      float error_pi, valor_PI, integral_pi, error_p;
7      float PIP_angulo;
8
```

```
9  void setup(){
10     Serial.begin(115200);
11         Wire.begin();
12   //Adjuntar motores, programar ESC's y habilitar sensores
13 }
14 void loop(){
15   //Medida ángulo
16     ...
17    angulo_x = 0.95*(angulo_x + dps_x*DT) + 0.05*angulo_acelerometro_x;
18
19   //PIP
20
21     error_pi = ANGULO_REFERENCIA - angulo_x;
22     integral_pi+=(error_pi*DT);
23     valor_PI = Kpi_p*error_pi + Kpi_i*integral_pi;
24     error_p = valor_PI - dps_x;
25
26     PIP_angulo = Kp_p*error_p;
27
28   //Control motores
29
30    velocidad_motor_izquierdo = Throttle_Motor_Izquierdo + PIP_angulo;
31      velocidad_motor_derecho   = Throttle_Motor_Derecho - PIP_angulo;
32
33      //El resto igual que en el PID
34 }
```

A partir de la línea 50 está la implementación del controlador, primero se calcula el error de la salida en grados y después de aplicarle a ese error el controlador proporcional integral se le resta la salida en grados por segundo. El resultado de la estabilización con este controlador para distintos valores de las constantes se muestra en la figura 5.10.



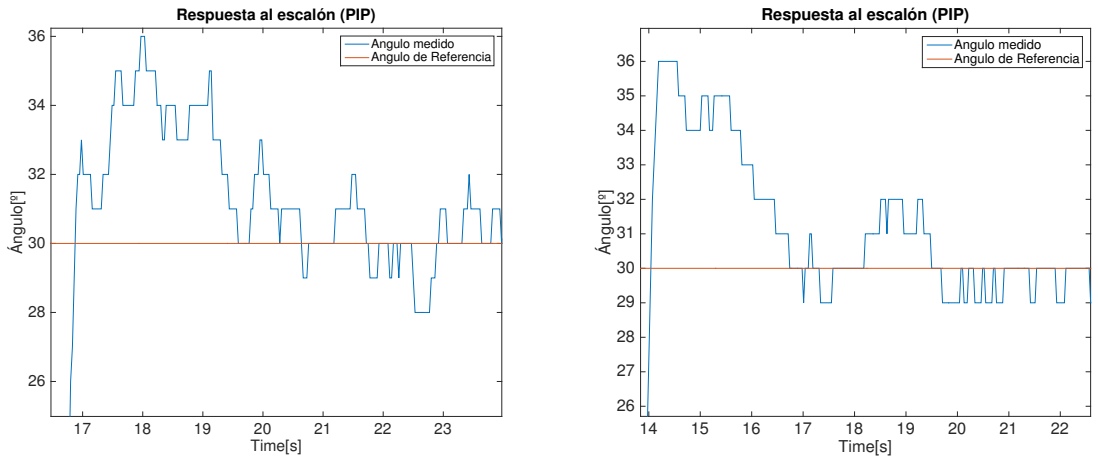
(a) Respuesta en la prueba 1.

(b) Respuesta en la prueba 2.

Figura 5.10: Respuesta del sistema para controlador PIP.

Los valores de las constantes para las gráficas de la figura 5.10 son $\mathbf{K_{pi\_p} = 5}$ $\mathbf{K_{pi\_i} = 2}$ y $\mathbf{K_{p\_p} = 4}$ para el **caso (a)** y $\mathbf{K_{pi\_p} = 3}$ $\mathbf{K_{pi\_i} = 2}$ y $\mathbf{K_{p\_p} = 2}$ para el **caso (b)**. A simple vista con este controlador se consigue un máximo sobreimpulso reducido y una alta velocidad de respuesta, sin embargo en el caso (b) el error en régimen permanente es elevado.

En la figura 5.10 se observa que cuando el brazo está estable en el régimen permanente previo al escalón el error es prácticamente nulo. Sin embargo, una vez se estabiliza en el ángulo de 30º el error es bastante más elevado. Esta diferencia se debe a que el peso del brazo no está totalmente compensado, si se coloca el brazo manualmente en una posición de 0º y se suelta el brazo va a caer hacia el lado derecho, es por ello que en función del lado hacia el que se incline el brazo la estabilización será distinta aunque el ángulo en valor absoluto sea el mismo. El error en régimen permanente después del escalón se muestra en detalle en la figura 5.11.



(a) Error en régimen permanente en la prueba 1. (b) Error en régimen permanente en la prueba 2.

Figura 5.11: Detalle del ruido presente en la respuesta en régimen permanente del controlador PIP.

Tabla 5.3: Parámetros en régimen transitorio PIP.

Parámetro	Caso (a)	Caso (b)
Tiempo de retardo (t_d)	0.93 seg	1.1 seg
Tiempo de subida (t_r)	1.07 seg	1.08 seg
Tiempo de pico (t_p)	2.19 seg	1.23 seg
Tiempo de establecimiento ($t_{s10\,\%}$)	10.95 seg	12.09 seg
Máximo sobreimpulso (M_p)	36º (1.12)	36º (1.2)

Con este controlador no se ha conseguido variar de forma significativa la respuesta del sistema mediante el ajuste de las constantes. La respuesta es demasiado oscilante y tarda mucho en llegar al régimen permanente, siendo un mal controlador si se busca precisión en el aparato a estabilizar. Las constantes de este tipo de controlador son más difíciles de ajustar que las del PID ya que se realimentan dos variables distintas: la velocidad angular en grados por segundo y la posición del brazo en grados. Al ajustar las constantes del PID se sabe que constante ajustar si se necesita mejorar una determinada característica del sistema, por ejemplo si se necesita mayor velocidad de respuesta se aumenta la constante K_d. Sin embargo en este controlador las pruebas se realizan experimentalmente, sin tener conocimiento previo del resultado que se obtendrá.

5.2.4 PIDPID

En este controlador vuelve a haber dos bucles de realimentación y a ambos errores se les aplica un controlador PID. El esquema de este regulador se muestra en la figura 5.12

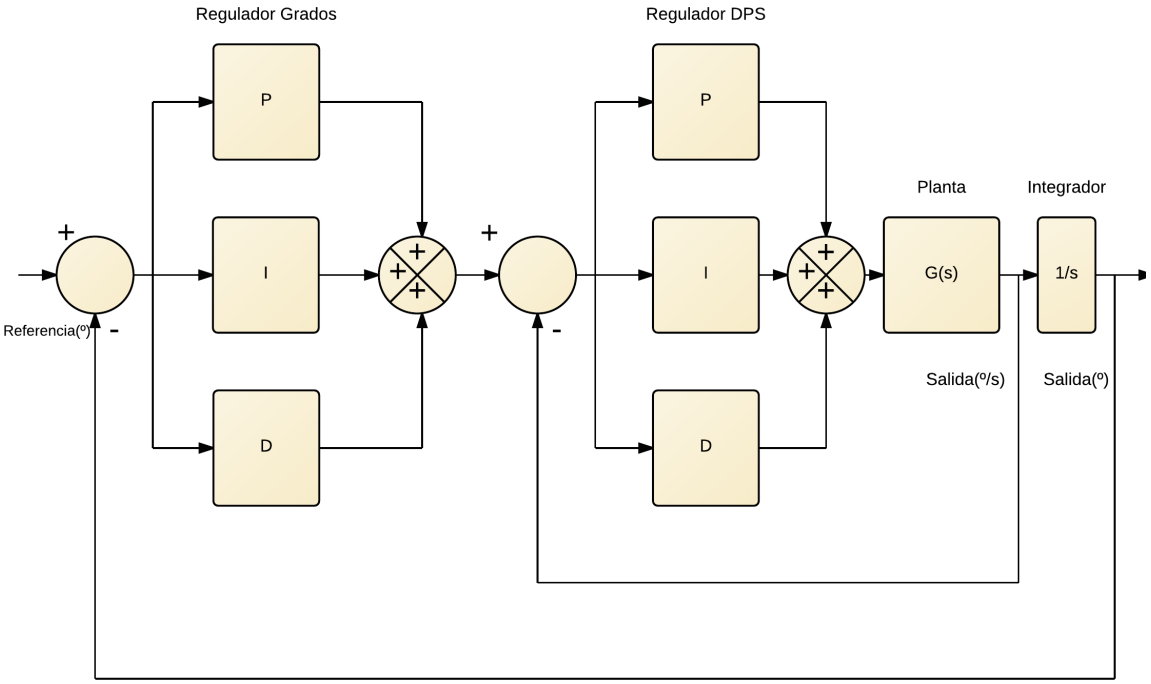


Figura 5.12: Esquema del controlador PIDPID.

La implementación de este controlador se realiza mediante el código 5.5.

Codigo 5.5: Implementación del PIDPID.

```
1  //Igual que en el PID
2
3  //VARIABLES PID
4
5      float Kp1 = 8, Kp2 = 8, Ki1 = 4, Ki2 = 5, Kd1 = 3, Kd2 = 1;
6      float error_PID1, error_pasado_PID1, error_PID2, error_pasado_PID2;
7      float derivativo_PID1, integral_PID1, derivativo_PID2, integral_PID2;
8      float PID1, PIDPID_angulo;
9
10 void setup(){
11   //Iniciar puerto serie, iniciar comunicación con la IMU, adjuntar motores... programar
12 }
13 void loop(){
14   //Medida ángulo
15
16    //PID del angulo
17
18         error_PID1 = ANGULO_REFERENCIA - angulo_x;
19         integral_PID1+=(error_PID1*DT);
20         derivativo_PID1 = (error_PID1-error_pasado_PID1)/DT;
21     PID1 = Kp1*error_PID1 + Ki1*integral_PID1 + Kd1*derivativo_PID1;
22
23     //PID de la velocidad angular
24
25         error_PID2 = PID1 - dps_x;
```

```
26          integral_PID2+=(error_PID2*DT);
27          derivativo_PID2 = (error_PID2-error_pasado_PID2)/DT;
28          PIDPID_angulo = Kp2*error_PID2 + Ki2*integral_PID2 + Kd2*derivativo_PID2;
29
30    //Control motores
31
32     velocidad_motor_izquierdo = Throttle_Motor_Izquierdo + PIDPID_angulo;
33       velocidad_motor_derecho   = Throttle_Motor_Derecho - PIDPID_angulo;
34     //El resto igual que en el PID
35          .....
36  }
```

El resultado de la estabilización con este controlador sólo se va a mostrar para un valor de las constantes, el ajuste de este controlador resulta compleja debido al elevado número de constantes que hay que variar.

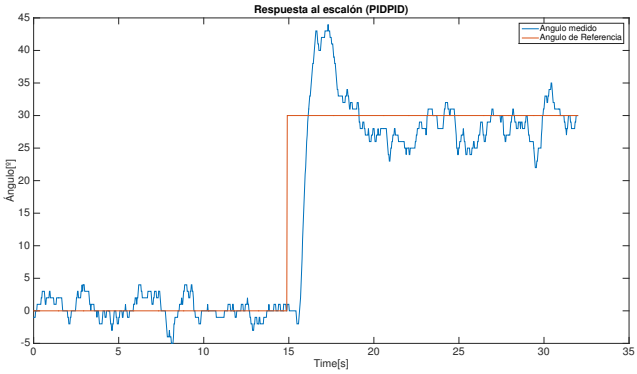


Figura 5.13: Respuesta del sistema para controlador PIDPID.

Los valores de las constantes para la gráfica de la figura 5.13 son $\mathbf{K_{p1} = 8}$ $\mathbf{K_{i1} = 4}$, $\mathbf{Kd1 = 3}$, $\mathbf{K_{p2} = 8}$ $\mathbf{K_{i2} = 5}$ y $\mathbf{K_{d2} = 1}$.

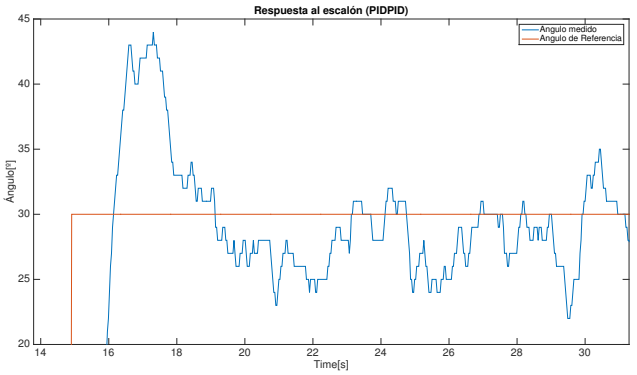


Figura 5.14: Detalle del ruido presente en la respuesta en régimen permanente del controlador PIDPID.

El error en régimen permanente es muy elevado debido al ruido introducido por los motores. Al haber tantas constantes por pequeño que sea el error el resultado del control siempre es elevado y los motores están constantemente oscilando entre su valor máximo y su valor mínimo, haciendo que el sistema tenga una presición muy baja.

Tabla 5.4: Parámetros en régimen transitorio PIDPID.

Parámetro	Valores
Tiempo de retardo (t_d)	1.12 seg
Tiempo de subida (t_r)	1.26 seg
Tiempo de pico (t_p)	2.43 seg
Tiempo de establecimiento ($t_{s10\,\%}$)	16.44 seg
Máximo sobreimpulso (M_p)	44º (1.46)

El aumento del número de constantes a ajustar hace que obtener buenos resultados con este controlador sea complicado y no sea una buena opción, pues con controladores menos complejos se obtienen mejores resultados con un menor tiempo de cómputo y una menor capacidad de cómputo, lo que es importante en una aplicación como esta en la que no se dispone de un procesador con mucha potencia.

5.2.5 Comparativa controladores

Habiendo implementado cada controlador de los arriba comentados y obtenido su respuesta ante un escalón se van a mostrar los mejores resultados de cada controlador para comprobar cuál de todos ellos es el más adecuado.

Tabla 5.5: Parámetros en régimen transitorio PIDPID.

Parámetro/Controlador	PID	PPI	PIP	PIDPID
Mejor caso	(b)	(a)	(a)	-
Tiempo de retardo (t_d)	1.2 seg	**0.83** seg	0.93 seg	1.12 seg
Tiempo de subida (t_r)	5.4 seg	**0.97 seg**	1.07 seg	1.26 seg
Tiempo de pico (t_p)	6.59 seg	**1.65 seg**	2.19 seg	2.43 seg
Tiempo de establecimiento ($t_{s10\,\%}$)	4.19 seg	**1.83 seg**	10.95 seg	16.44 seg
Máximo sobreimpulso (M_p)	**31º (1.03)**	35º (1.16)	36º (1.12)	44º (1.46)
Error máximo en rég. permanente	**1º**	2º	2º	7º

En el caso del controlador PID el tiempo de pico es superior al tiempo de establecimiento porque el sistema es subamortiguado, es decir, la salida no llega a superar el ángulo de referencia durante el régimen transitorio, pero sí en algunos momentos del régimen permanente, como se puede observar en la figura 5.4. La elección del controlador más adecuado depende de la aplicación para la que se necesite, si se necesita buena **precisión** el mejor controlador es el **PID**, su error en régimen permanente es pequeño y no tiene sobreimpulso en régimen transitorio. Si lo que se necesita es alta **velocidad de respuesta** el controlador más adecuado es el **PPI**, con él se obtiene una respuesta el doble de rápida que con el segundo controlador más rápido, siendo el régimen transitorio muy corto. Como ya se ha comentado en la sección 5.2.4, el controlador PIDPID no es una buena opción pues su sintonización es complicada por tener el doble de constantes a establecer que el resto de controladores.

5.2.6 Eliminación previa de errores

En este apartado se van a comentar dos errores que han surgido durante las pruebas realizadas con el brazo. Debido a estas pruebas se han encontrado maneras de conseguir que el sistema se vuelva inestable, sin embargo con unas pocas líneas de código se pueden eliminar estos errores.

5.2.6.1 Anti-Windup

Como se explica en [15] el windup es un fenómeno que ocurre cuando la salida de un controlador proporcional integral está saturado, resultando en mal funcionamiento e incluso inestabilidad. Al controlar la velocidad de un motor, un cambio brusco en la velocidad puede provocar que la corriente generada por el controlador electrónico de velocidad llegue a su valor máximo, que viene determinado por el protector del controlador, la saturación magnética y el sobrecalentamiento del motor.

Este efecto se ha conseguido durante las pruebas sujetando el brazo en una posición alejada de la posición de referencia, provocando así que el factor integral del PID esté sumando constantemente y se acumule durante unas decenas de segundos. Al volver a soltar el brazo la suma del integral es muy elevada y el error a corregir por el controlador es mayor que el real, haciendo que l brazo no vuelva a estabilizarse hasta que el integral tiene un valor pequeño. Esto puede pasar en un caso real con una fuerza externa como por ejemplo el viento, si hay una fuerte ráfaga de viento y provoca que el aparato salga de la posición de referencia se produce este efecto. Para eliminar este fenómeno sólo hay que limitar el valor del parámetro integral del controlador, como se muestra en el código 5.6. El código mostrado corresponde al controlador PID, pero se aplica al resto de controladores que incluyan un integral de la misma manera. A este controlador se le puede denominar controlador Anti-Windup, pues no le afecta este fenómeno.

Codigo 5.6: Anti-Windup.

```
1  #define MAXIMO_INTEGRAL 200
2  ...
3  void loop(){
4
5    //PID
6
7     error = ANGULO_REFERENCIA - angulo_x;
8         integral+=(error*DT);
9         derivativo = (error-error_pasado)/DT;
10        PID_angulo = Kp*error + Ki*integral + Kd*derivativo;
11
12
13      if(integral > MAXIMO_INTEGRAL)
14          integral = MAXIMO_INTEGRAL;
15      else if(integral < -MAXIMO_INTEGRAL)
16          integral = -MAXIMO_INTEGRAL;
17   ...
18 }
```

De la línea 15 a la línea 18 se observa la implementación del Anti-Windup. Hay que limitarlo para todo el margen de valores de la variable integral, puesto que la variable error puede ser positiva o negativa se limita para valores positivos y para valores negativos. El valor de *MAXIMO_INTEGRAL* se ha establecido mediante pruebas, comprobando que con valores mayores se sigue produciendo el fenómeno wind-up y con valores menores se empeora la estabilización del brazo.

5.2.6.2 Limitación de la velocidad de los motores

La velocidad de los motores se puede variar mediante el ESC desde un mínimo de una señal de 1 ms hasta un máximo de una señal de 2 ms para que estén girando y no se paren. Sin embargo, la fuente de alimentación disponible entrega una corriente limitada y reducida. Debido a la forma de implementar el control de los motores si no se limita su velocidad y el error es elevado se puede dar la situación de que uno de los motores gire a un valor por debajo del mínimo y se pare y el otro gire a una velocidad mayor que la máxima y pueda llegar a romperse. Además se ha comprobado mediante las pruebas que si uno de los motores gira a máxima velocidad y el otro a mínima velocidad la fuente no es capaz de entregar la corriente necesaria y los motores no giran a la velocidad adecuada para estabilizar el brazo, provocando un mal funcionamiento. Para solucionar este problema se debe limitar la velocidad de los motores antes de mandar la señal al ESC, como se muestra en el código 5.7.

Codigo 5.7: Limitación de la velocidad de los motores.

```

...
#define MAX_VEL                   1500
#define MIN_VEL                   1000

#define MOTOR_IZQUIERDO_PIN       9
#define MOTOR_DERECHO_PIN         10

#define Throttle_Motor_Izquierdo  1300   //Velocidad de los motores si el error es 0
#define Throttle_Motor_Derecho    1300

double angulo_y

//VARIABLES MOTORES

Servo motorizquierdo, motorderecho;
int velocidad_motor_izquierdo, velocidad_motor_derecho;

...


void loop()
{
  ...

  //PID

   error = ANGULO_REFERENCIA - angulo_x;
       integral+=(error*DT);
       derivativo = (error-error_pasado)/DT;
       PID_angulo = Kp*error + Ki*integral + Kd*derivativo;

  //Control motores

     velocidad_motor_izquierdo = Throttle_Motor_Izquierdo + PID_angulo;
     velocidad_motor_derecho   = Throttle_Motor_Derecho - PID_angulo;

      velocidad_motor_izquierdo = (velocidad_motor_izquierdo > MAX_VEL) ? MAX_VEL:
          velocidad_motor_izquierdo;
     velocidad_motor_izquierdo = (velocidad_motor_izquierdo < MIN_VEL) ? MIN_VEL:
         velocidad_motor_izquierdo;
```

```
40
41      velocidad_motor_derecho = (velocidad_motor_derecho > MAX_VEL) ? MAX_VEL:
            velocidad_motor_derecho;
42      velocidad_motor_derecho = (velocidad_motor_derecho < MIN_VEL) ? MIN_VEL:
            velocidad_motor_derecho;
43
44      motorizquierdo.writeMicroseconds(velocidad_motor_izquierdo);
45      motorderecho.writeMicroseconds(velocidad_motor_derecho);
46
47      error_pasado = error;  //Para calcular el derivativo
48  }
```

De la línea 38 a la línea 43 se muestra cómo se limita la velocidad de los motores. Si no se limita su valor la velocidad de los motores asignada en las líneas 35 y 36 pueden salirse de los valores mínimo y máximo y el sistema dejaría de ser estable.

CAPITULO VI

Manejo del brazo mediante LabView

Capítulo 6

Manejo del brazo mediante LabView

6.1 Introducción

Hasta esta parte del trabajo se han explicado todos los pasos necesarios para conseguir una buena estabilización del brazo, desde cómo obtener una estimación precisa del ángulo hasta cómo implementar los controladores para estabilizar el brazo en la posición deseada. Para poder realizar pruebas es necesario hacer un programa mediante el que se pueda variar el ángulo de referencia y las constantes de los controladores. Esto se puede hacer mandando comandos mediante el puerto serie del Arduino y mostrándolos en el monitor serie disponible en la IDE de Arduino pero es poco intuitiva y hay que actualizar los datos mostrados por pantalla a baja frecuencia para que sean visibles. Existe una aplicación de National Instruments [16] para ciencia e ingeniería, llamada LabView, que es un entorno de desarrollo gráfico que permite controlar sistemas de una forma más visual y reducir el tiempo empleado para las pruebas. En este apartado se va a explicar brevemente los pasos necesarios para desarrollar un programa en LabView que muestre los resultados por pantalla y recopile datos para después obtener las gráficas en Matlab.

6.2 Programación del Arduino

El primer paso es programar el Arduino las entradas que se van a usar para variar los valores del ángulo de referencia y de las constantes de los controladores. La solución implementada es leer el puerto serie y asociar ciertos caracteres incrementos o decrementos de los valores, como se muestra en el código 6.1.

Codigo 6.1: Programación en Arduino para variar referencia y constantes.

```
1  #define ANGULO_REF_MAX  70
2  #define ANGULO_REF_MIN -70
3  float INCREMENTO_ANGULO_POSITIVO = 1;
4  float INCREMENTO_ANGULO_NEGATIVO = -1;
5  char val;
6
7  void serialEvent(){
8
9    if( Serial.available() )    // si hay dato y el puerto lo lee
10       val = Serial.read();    // lee y almacena el dato en 'val'
11
12     switch (val){
13
```

```
14        case '+':
15          ANGULO_REFERENCIA += INCREMENTO_ANGULO_POSITIVO;
16          if (ANGULO_REFERENCIA > ANGULO_REF_MAX)
17                ANGULO_REFERENCIA -= INCREMENTO_ANGULO_POSITIVO;
18        break;
19
20        case '-':
21          ANGULO_REFERENCIA += INCREMENTO_ANGULO_NEGATIVO;
22          if (ANGULO_REFERENCIA < ANGULO_REF_MIN)
23                ANGULO_REFERENCIA -= INCREMENTO_ANGULO_NEGATIVO;
24        break;
25
26        case 'R':
27          INCREMENTO_ANGULO_POSITIVO+=1;
28          INCREMENTO_ANGULO_NEGATIVO-=1;
29        break;
30
31        case 'r':
32          INCREMENTO_ANGULO_POSITIVO-=1;
33          INCREMENTO_ANGULO_NEGATIVO+=1;
34        break;
35
36       case 'd':
37          Kd-=1;
38        break;
39
40        case 'D':
41          Kd+=1;
42        break;
43
44        case 'i':
45          Ki-=1;
46        break;
47
48        case 'I':
49          Ki+=1;
50        break;
51
52        case 'p':
53          Kp-=1;
54        break;
55
56        case 'P':
57          Kp+=1;
58        break;
59
60      }
61  }
```

La limitación del ángulo se implementa por seguridad por código además de físicamente con la barra horizontal entre las dos vigas que sujetan el brazo. La función *serialEvent()* es un evento que ocurre cuando un dato llega al puerto serie a través de RX. El Arduino lo comprueba cada iteración del bucle principal. Para cada acción se ha asociado un carácter en minúsculas y el mismo carácter en mayúsculas. Por ejemplo, si se pulsa D se incrementa en uno la constante K_d . En el caso de pulsar R lo que se hace es variar el tamaño del escalón de referencia que se incrementará o decrementará al pulsar + o -, es decir, si se pulsa R 4 veces la variable *INCREMENTO_ANGULO_POSITIVO* valdrá 5 y la próxima vez que se pulse + el ángulo de referencia será 5º mayor.

Mediante el código 6.1 se ha mostrado cómo implementar la entrada de datos, pero también es necesario mostrar los resultados por pantalla. La única manera de comunicación entre la aplicación que se desarrollará en LabView y el programa cargado en Arduino es el puerto serie, por lo tanto para mostrar los resultados hay que escribir desde el programa de Arduino en el puerto serie los valores y leerlos en LabView.

Codigo 6.2: Escritura de datos

```
1
2  void Imprimir_LabView(){
3
4     Serial.print((int) angulo_x);
5     Serial.print("\t");
6
7     Serial.print((int) velocidad_motor_izquierdo);
8     Serial.print("\t");
9
10    Serial.print((int) velocidad_motor_derecho);
11    Serial.print("\t\n");
12 }
```

Debido a la gran cantidad de código que se va acumulando para poder estabilizar el brazo es conveniente organizarlo en pestañas, disponibles en la IDE del Arduino. En cada pestaña se pueden programar las funciones asociadas a cada parte del trabajo, una para el cálculo del PID, otra para la estimación del ángulo...

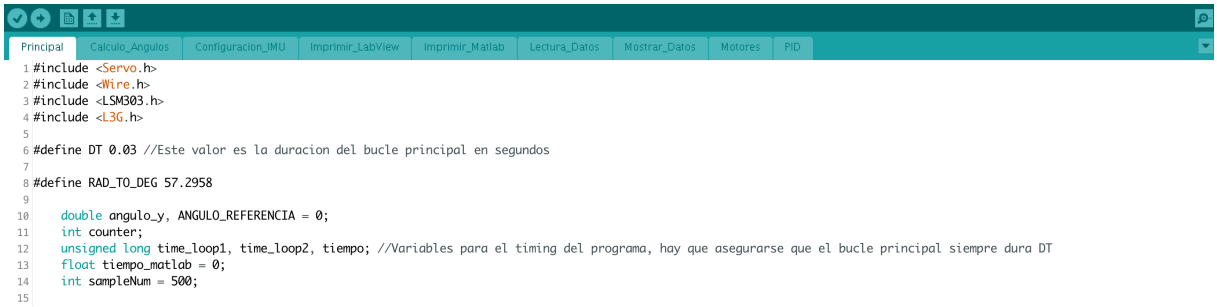
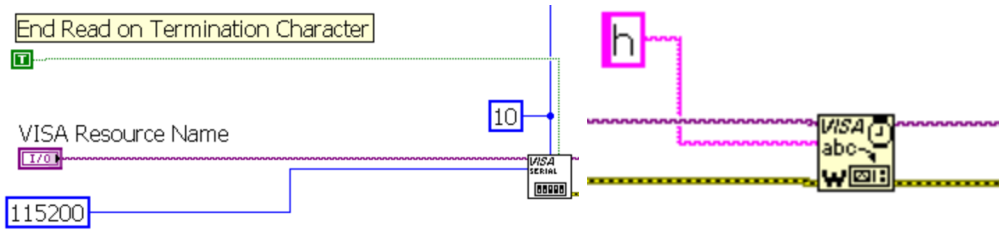


Figura 6.1: Distribución en pestañas del programa implementado para el controlador PID.

6.3 Desarrollo en LabView

Habiendo programado los caracteres que realizarán las modificaciones sólo queda realizar el programa en LabView. Para poder ejecutar el programa desde LabView se debe abrir una comunicación serie igual que si se hiciese mediante Arduino. Como se indica en la página de National Instruments [17], para tener acceso al puerto serie usando LabView se debe iniciar una sesión VISA y configurarla con los mismos parámetros que en el Arduino, el baud rate y el carácter de fin de lectura, que en Arduino viene por defecto como 0xA, que es el retorno de carro (\n). Una vez configurada la sesión VISA empezará la ejecución del programa cargado en el Arduino, por lo tanto hay que programar los mismos pasos seguidos en Arduino desde LabView, que son escribir un carácter cualquiera en el puerto serie de Arduino para la calibración del ESC y después comienza el bucle principal.



(a) Configuración sesión VISA. (b) Escritura de carácter.

Figura 6.2: Pasos para que comience el bucle principal.

Ya se puede pasar al bucle principal para modificar los valores de las constantes y del ángulo de referencia. La duración del bucle de LabView tiene que ser la misma que la del bucle principal del Arduino, tienen que ser síncronos. En LabView se debe estar leyendo el puerto serie cada iteración para mostrar los resultados por pantalla y se deben programar botones para realizar las modificaciones, todo ello metido dentro de un bucle *while* que acabe mediante un botón de Stop. Una captura de la interfaz implementada se muestra en la figura 6.3.

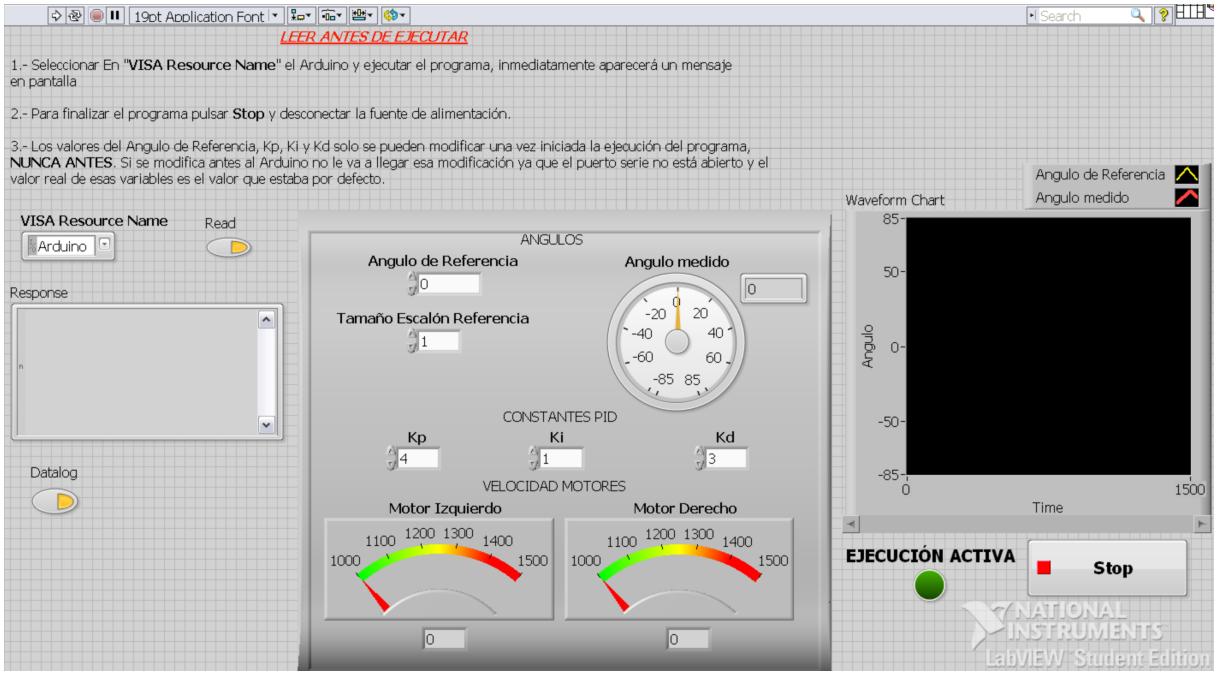


Figura 6.3: Interfaz gráfica desarrollada en LabView.

Los elementos que se han programado en la interfaz son:

- **Botón para modificar el ángulo de referencia**: para programarlo hay que enviar el carácter correspondiente por el puerto serie, que es + para los incrementos y - para los decrementos.
- **Botón para modificar el tamaño del escalón del ángulo de referencia**: se debe escribir por el puerto serie el carácter correspondiente, que es R para incrementos y r para decrementos.
- **Botones para modificar las constantes del controlador**: estos controles son muy útiles para modificar el tipo de respuesta del controlador sin tener que recurrir a la ejecución del programa cada vez que se quiera modificar el valor de estas constantes.

- **Botón para activar/desactivar el *datalog***: el *datalog* consiste en escribir datos en un fichero para su posterior análisis. En este trabajo se escriben en un fichero *.txt* el ángulo medido, el ángulo de referencia y el instante de tiempo para cada muestra con el fin de obtener una representación en Matlab.

- **Indicadores de la velocidad de los motores**: ya que en la sección 6.2 se ha mostrado cómo programar en Arduino la escritura por el puerto serie de los valores de los motores, en LabView se debe leer el puerto serie cada iteración del bucle y se tiene el valor de la velocidad de cada motor.

- **Indicador del ángulo medido**: también llega por el puerto serie, por lo tanto se debe leer el puerto serie y asignar el valor leído al indicador.

- **Gráfica que representa en tiempo real el ángulo medido y el ángulo de referencia**: mediante esta gráfica se observa el tipo de respuesta del sistema en tiempo real, aunque para posteriores análisis como tiempo de establecimiento, máximo sobreimpulso, etc., se usará Matlab es útil para modificar los valores de las constantes hasta conseguir la respuesta deseada.

Para finalizar con este apartado se va a mostrar una secuencia de imágenes en las que se ha ejecutado el programa con el controlador PID variando el ángulo de referencia, el tamaño del escalón del ángulo de referencia y, por último, las constantes del controlador. El orden de la secuencia de imágenes de la figura 6.4 es de izquierda a derecha y de arriba a abajo.

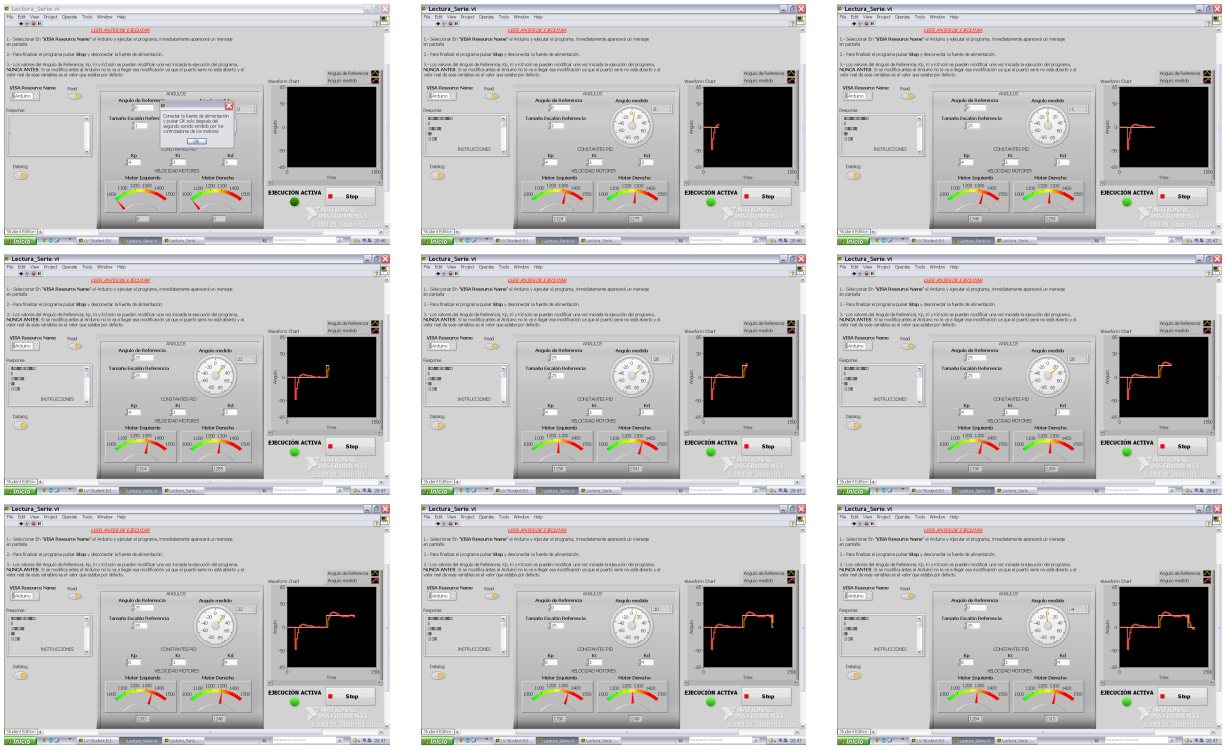


Figura 6.4: Secuencia de imágenes de la ejecución del programa con el controlador PID.

CAPITULO VII

Conclusiones y trabajo futuro

Capítulo 7

Conclusiones y trabajo futuro

7.1 Conclusiones

El trabajo llevado a cabo ha requerido conocimientos de diversas ramas, utilizando conceptos ya estudiados a lo largo de la carrera y teniendo que realizar un estudio de algunos temas a partir de otros artículos de investigación y libros. En la figura 7.1 se muestra un diagrama de los pasos llevados a cabo para realizar el proyecto.

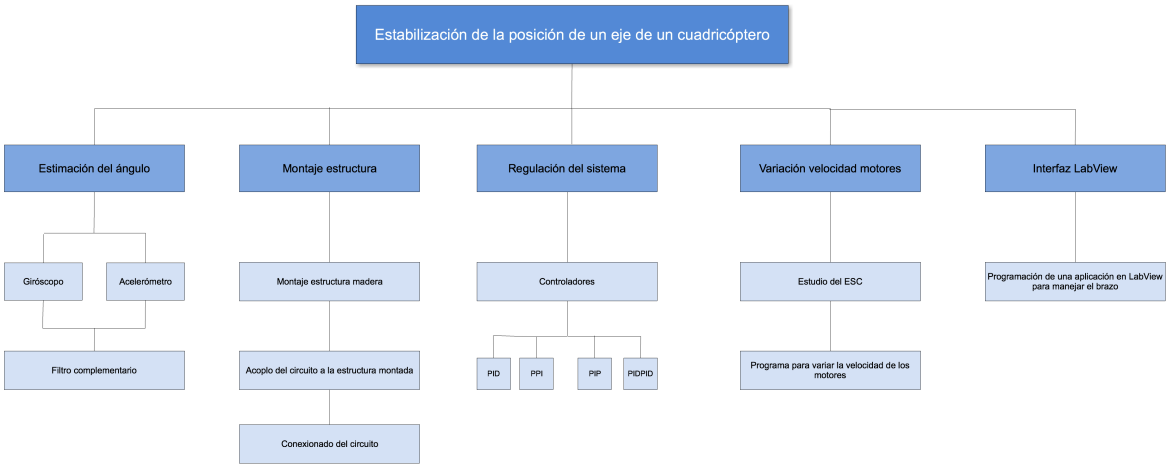


Figura 7.1: Diagrama del sistema.

Para llevar a cabo este proyecto se han hecho varias tareas de diversos ámbitos:

- **Programas para la estimación del ángulo**: ha sido necesario desarrollar programas propios para poder obtener una estimación del ángulo de los sensores utilizados, algunos algoritmos se han tomado como referencia de la bibliografía pero el programa ha sido desarrollado desde cero.
- **Programa para regular la velocidad de los motores**: también se han hecho programas para variar la velocidad de los motores mediante el ESC.
- **Controladores**: cada controlador ha requerido un programa independiente y se han realizado pruebas con cada uno de ellos.

- **Scripts Matlab**: ha sido necesario usar esta herramienta para obtener gráficas de los resultados de la respuesta del sistema.
- **Interfaz LabView**: el desarrollo de una aplicación para manejar el brazo y realizar pruebas con él es imprescindible, una buena opción es realizar esta aplicación con LabView. El aprendizaje para la programación en LabView ha sido rápido y los resultados son muy buenos.
- **Montaje de la estructura**: partiendo de un tablero de madera se han cortado las piezas necesarias para montar la estructura y se ha ensamblado obteniendo la estructura mostrada en la figura 2.11.
- **Conexionado del circuito**: el sistema constaba de dos circuitos independientes, uno de potencia que alimentaba los motores mediante la batería, y uno de control mediante el que se mandaban las señales de control a los motores y se leían los datos de los sensores.

Con el desarrollo de este trabajo se ha podido comprobar cómo un buen ajuste de las constantes de los controladores puede hacer que el control sobre el sistema pase de tener una baja precisión y baja velocidad de respuesta a ser una respuesta rápida y precisa. Con los controladores implementados se puede abordar la construcción y el control de un cuadricóptero aplicando el mismo controlador a cada eje.

7.2 Trabajo futuro

En este apartado se van a comentar algunas mejoras que se pueden hacer a partir del trabajo realizado en este TFG.

- **Identificación del modelo de la planta**: si se obtiene el modelo de la planta se pueden realizar simulaciones con la herramienta *Simulink* de Matlab y se puede realizar una mejor acción de control mediante la obtención de la respuesta en frecuencia, del lugar de las raíces...
- **Añadir un segundo eje y eliminar las barras verticales**: si se eliminan las barras verticales y se añade un segundo eje perpendicular se construye un cuadricóptero, siendo más interesante puesto que se puede controlar el vuelo.
- **Manejo mediante radiofrecuencia**: si se le añade un receptor/emisor de radiofrecuencia al aparato se puede controlar el ángulo mediante un mando a distancia de forma inalámbrica, este requisito se hace imprescindible si se quiere construir un cuadricóptero.
- **Manejo mediante Wi-Fi**: si se añade un módulo Wi-Fi a la placa se puede crear una red mediante la que controlar inalámbricamente el brazo desde el ordenador, sin necesidad de tener conectado el cable USB todo el tiempo que se quieran realizar pruebas.

CAPITULO VIII

Diagramas

Capítulo 8

Diagramas

8.0.1 Diagrama de flujo del sistema

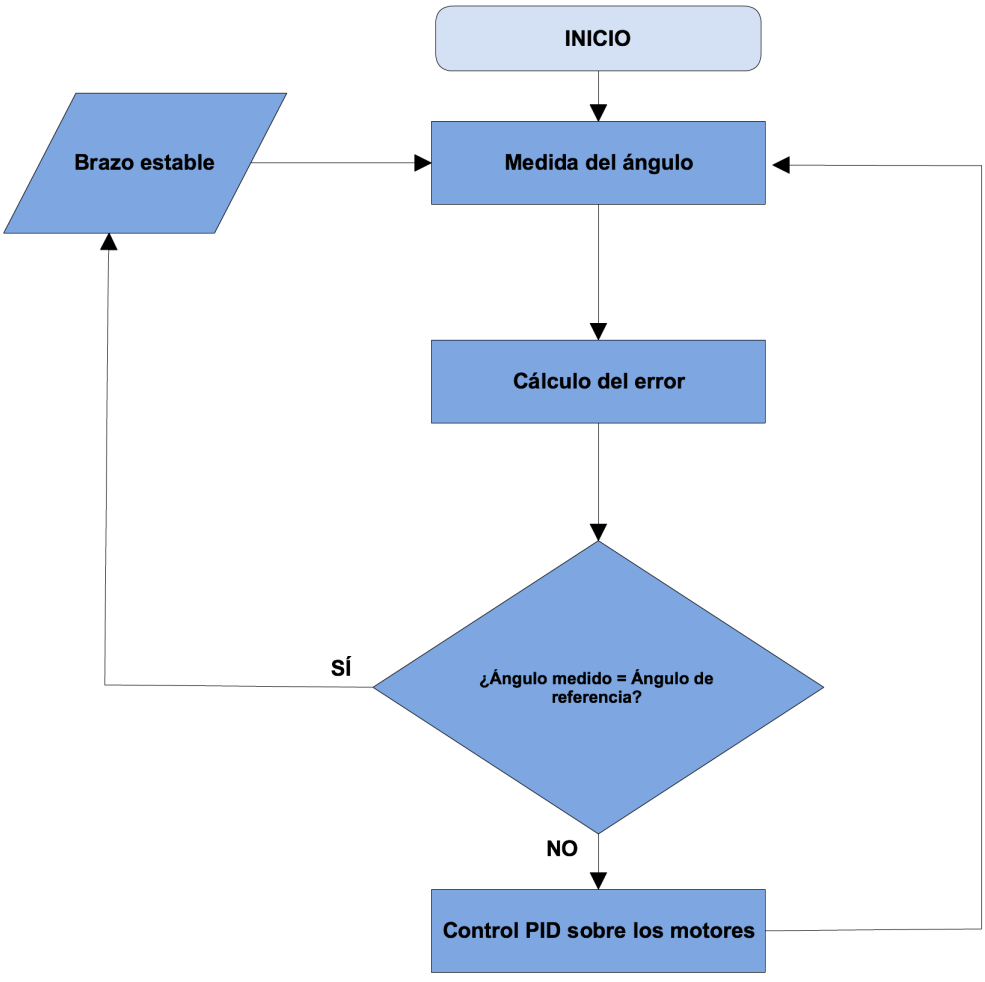


Figura 8.1: Diagrama de flujo del sistema.

8.0.2 Diagrama general del sistema

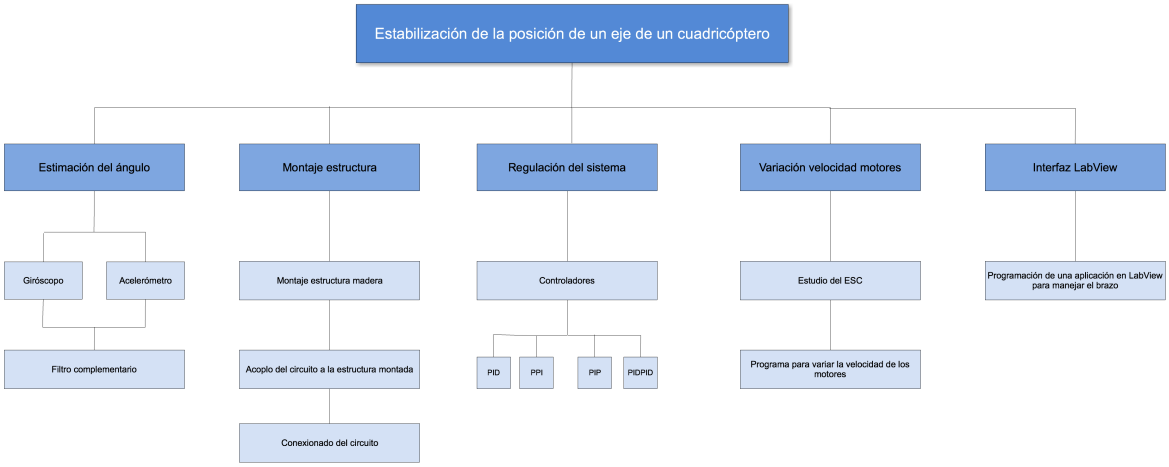


Figura 8.2: Diagrama general del sistema.

8.0.3 Esquema de la conexión de los motores y el sensor

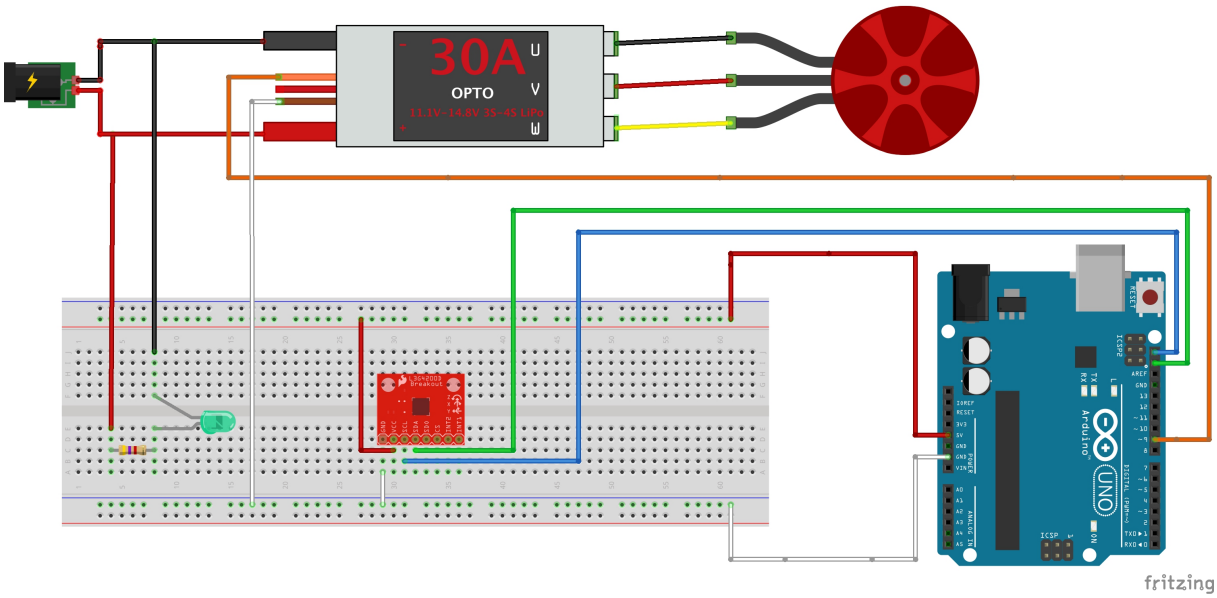


Figura 8.3: Esquema de la conexión de un motor.

8.0.4 Diagramas de bloques de los controladores

8.0.4.1 PID

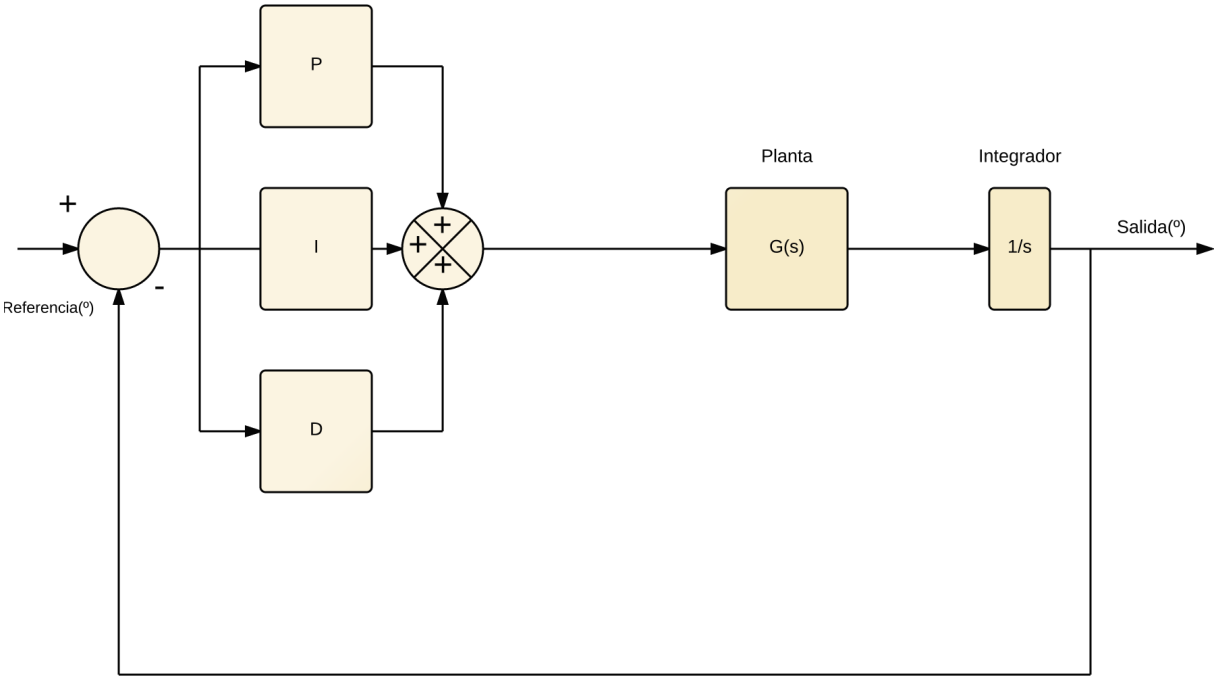


Figura 8.4: Diagrama de bloques del controlador PID.

8.0.4.2 PPI

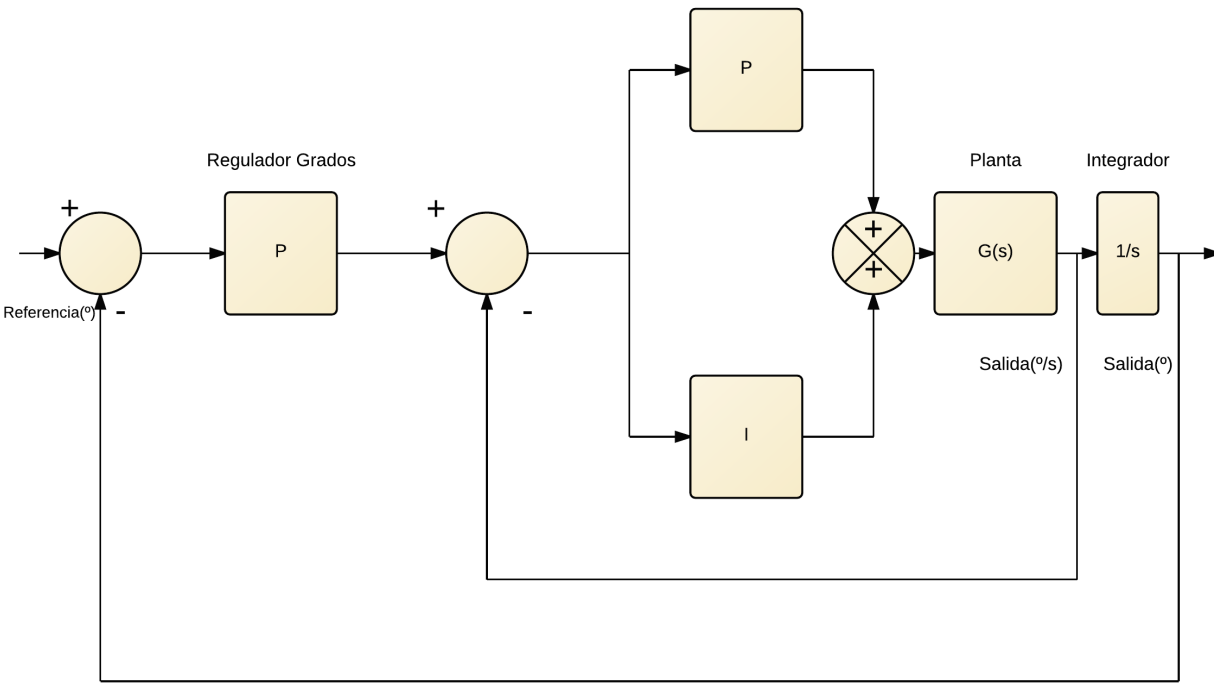


Figura 8.5: Diagrama de bloques del controlador PPI.

8.0.4.3 PIP

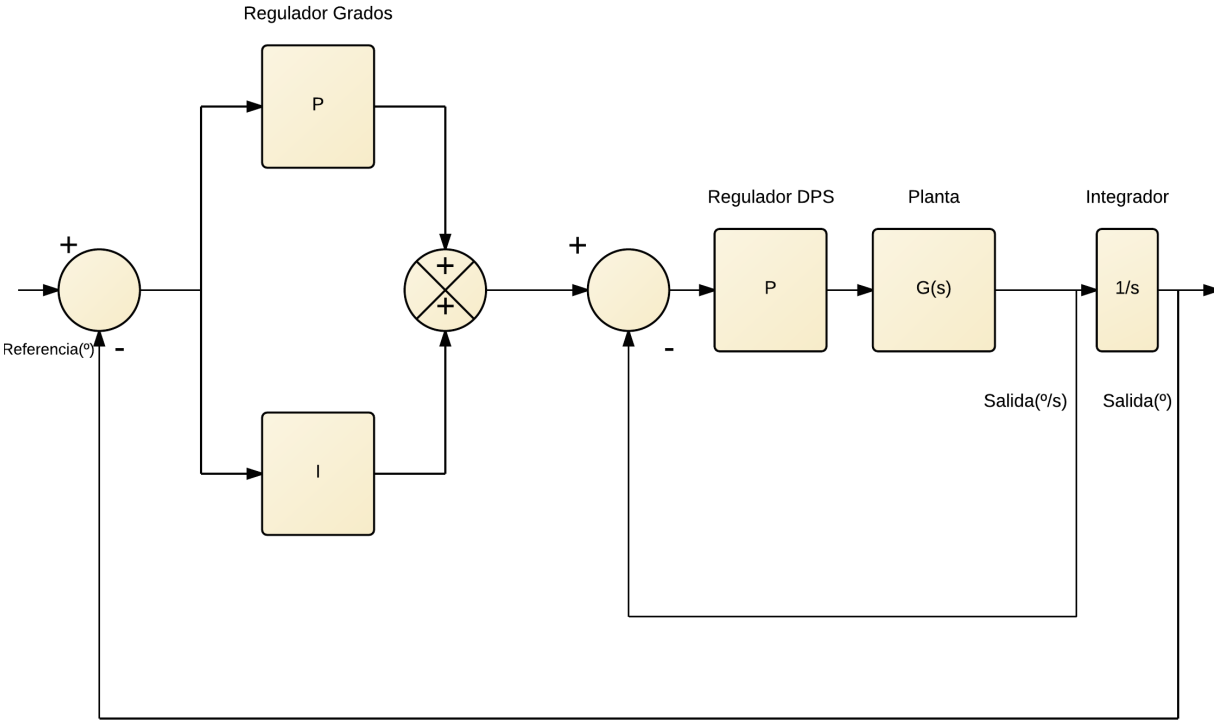


Figura 8.6: Diagrama de bloques del controlador PIP.

8.0.4.4 PIDPID

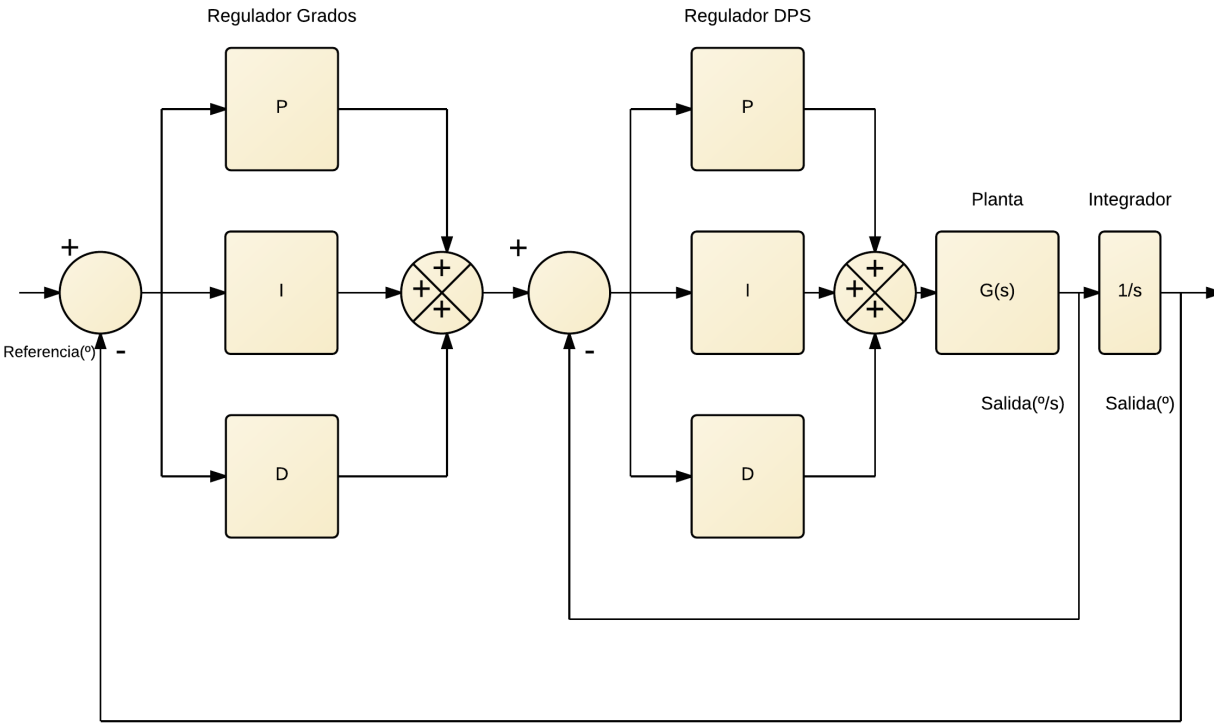


Figura 8.7: Diagrama de bloques del controlador PIDPID.

8.0.5 Diagramas de la estructura

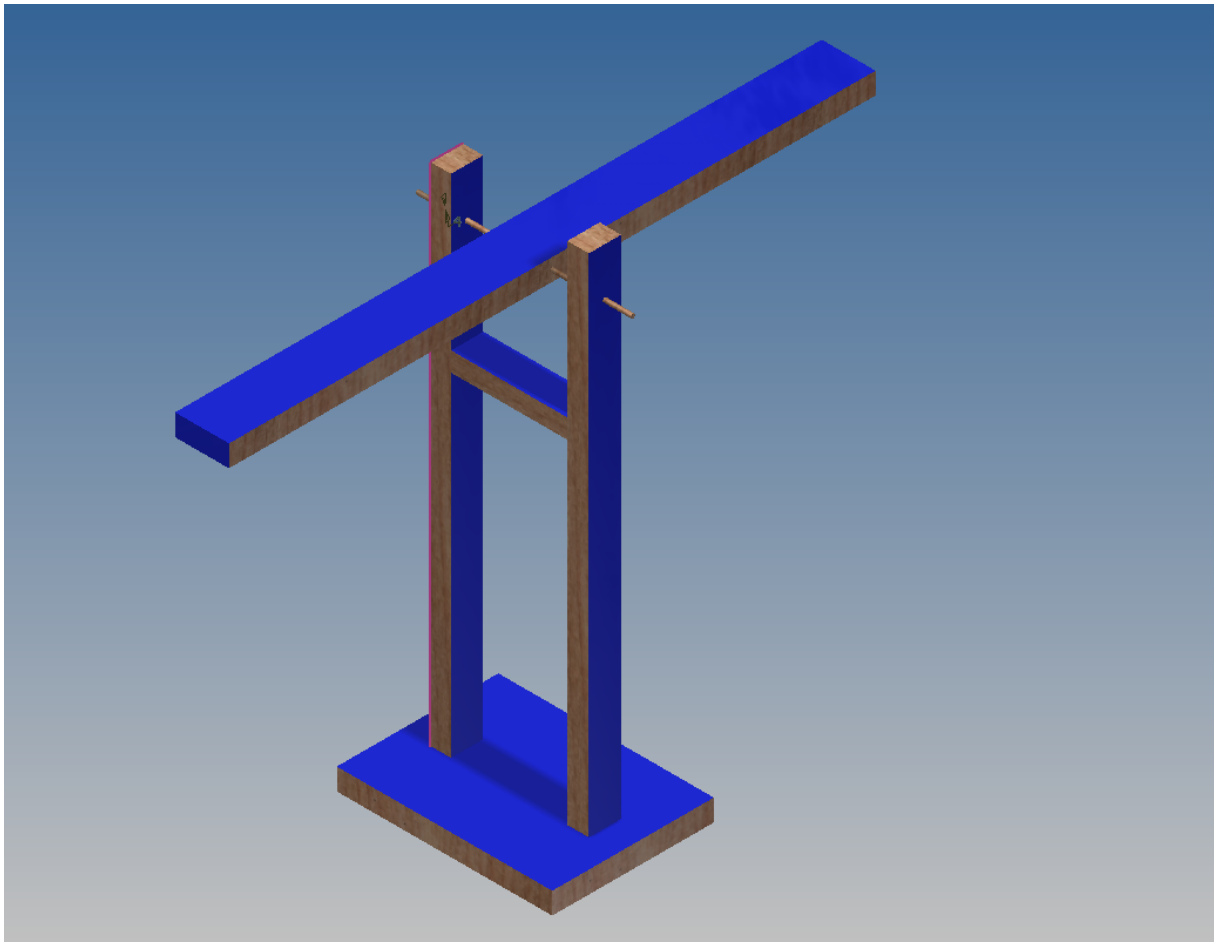


Figura 8.8: Representación del objeto en 3D con Autodesk Inventor.

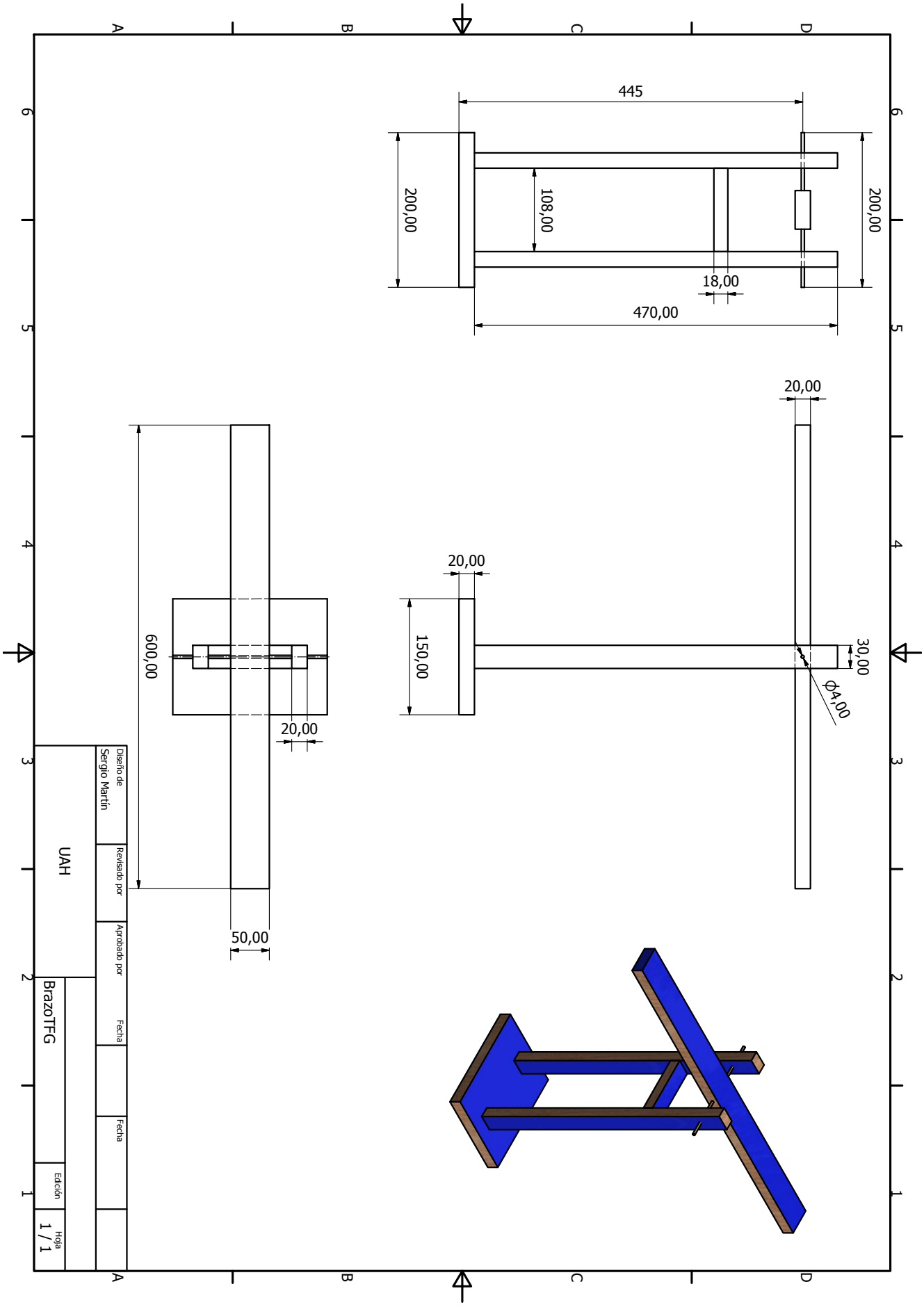


Figura 8.9: Vistas de la estructura.

CAPITULO IX

Pliego de condiciones

Capítulo 9

Pliego de condiciones

9.0.6 Requisitos de Hardware

- **Microcontrolador Arduino Uno**
- **Unidad de de medición inercial (IMU) AltIMU-10**
- **PC con puerto USB**
- **Fuente de alimentación**
- **Motores Brushless**
- **Hélices**
- **Controladores electrónicos de velocidad (ESC)**
- **Tablones de madera para ensamblar la estructura**
- **Placa de prototipos para soldar el circuito**

9.0.7 Requisitos de Software

- **Matlab R2015 versión estudiante**
- **IDE[1] de Arduino**
- **Sistema operativo compatible con la IDE de Arduino**
- **LabView versión estudiante**
- **AutoDesk Inventor versión estudiante**

[1] Siglas de Integrated Development environment, en español entorno de desarrollo integrado

CAPITULO X

Presupuesto

Capítulo 10

Presupuesto

10.0.8 Coste del Hardware

Tabla 10.1: Costes Hardware (21 % IVA incluido).

CONCEPTO	Precio [€/ud]	Unidades	TOTAL[€]
Arduino Uno	24.2	1	24.2
AltIMU-10	24.97	1	24.97
MacBook Pro	1499	1	1499
Fuente Octek006	39.95	1	39.95
Motor Brushless	5.79	2	11.58
ESC	6.90	2	13.80
Hélice	4.97	2	9.95
Tablero madera	10	1	10
Placa prototipos	10	1	10
TOTAL HARDWARE			**1643.45**

10.0.9 Coste del Software

Tabla 10.2: Costes Software (21 % IVA incluido).

CONCEPTO	Precio [€/ud]	Unidades	TOTAL[€]
IDE de Arduino	0	1	0
Matlab estudiante	49.99	1	49.99
LabView estudiante	19.99	1	19.99
Inventor estudiante	19.99	1	19.99
Mac OS X	0	1	0
TexMaker	0	1	0
TOTAL SOFTWARE			**89.97**

10.0.10 Importe total

Tabla 10.3: Coste total.

CONCEPTO	COSTE[€]
Hardware	4643.5
Software	89.97
TOTAL	**1733.42**

CAPITULO XI

Manual de usuario

Capítulo 11

Manual de usuario

11.1 Introducción

En este apartado se van a especificar todos los pasos necesarios para poder ejecutar el programa final de LabView de manejo del brazo y estabilizarlo en el ángulo deseado.

11.1.1 Subir programa al Arduino

El primer paso para poder ejecutar el programa es conectar el Arduino mediante el puerto USB al ordenador y subir el programa que se quiere ejecutar. La carga del programa se hace mediante la IDE de Arduino, disponible en [18], pulsando el botón que se indica en la figura 11.1.

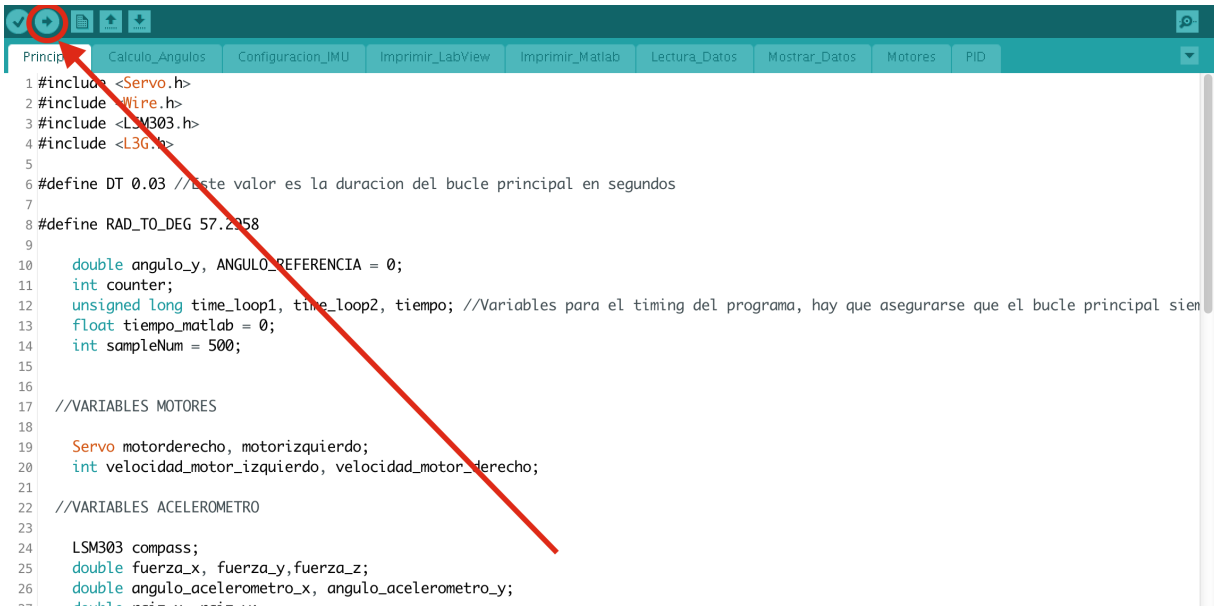


Figura 11.1: Cargar programa en el Arduino.

11.1.2 Comenzar ejecución

Con el programa ya cargado en el Arduino ahora se debe comenzar la ejecución del programa. Previo a la ejecución es importante leer las consideraciones del programa escritas en la interfaz para conocer su funcionamiento. Estas consideraciones son:

1. **Seleccionar en la sesión VISA el Arduino**: normalmente está seleccionado por defecto pero hay que asegurarse previo a la ejecución de que existe comunicación entre LabView y Arduino.
2. **Para finalizar el programa pulsar *Stop* y desconectar la fuente de alimentación**: si sólo se pulsa *Stop* el brazo se queda estabilizado en la última posición indicada por el usuario, para que se paren los motores hay que desconectar la fuente de alimentación.
3. **Los valores del ángulo de referencia y de las constantes del controlador no pueden cambiarse antes de la ejecución del programa**: debido a la forma de comunicación entre LabView y Arduino hasta que no se ejecuta el programa en LabView no comienza la sesión VISA y no existe comunicación entre la aplicación y Arduino; los valores por defecto coinciden en el programa cargado en Arduino y en la aplicación de LabView, por lo tanto si se cambian en LabView antes de comenzar la ejecución del programa los valores van a ser distintos de los que había por defecto y va a haber un mal funcionamiento.

Para comenzar la ejecución del programa en la ventana principal de LabView se debe pulsar el botón ejecutar como se muestra en la figura 11.2.

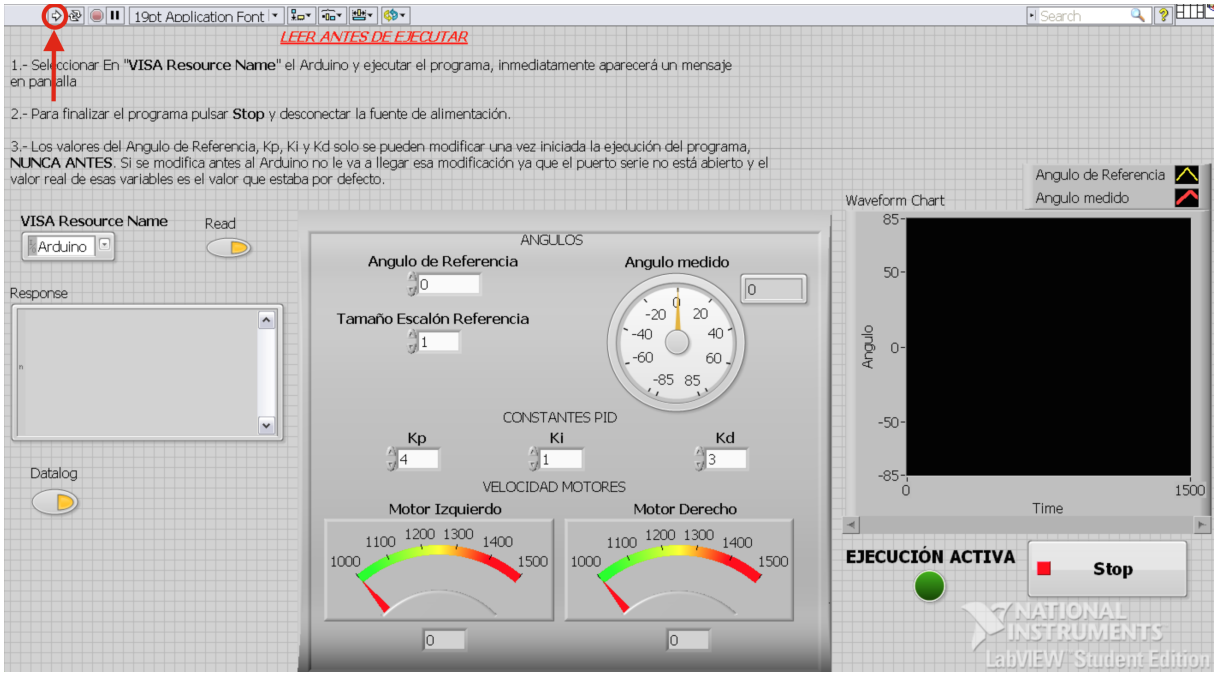


Figura 11.2: Comenzar ejecución.

Tras este paso ya ha comenzado la ejecución del programa y se ha establecido comunicación entre la aplicación de LabView y el Arduino, pues ya se ha abierto el puerto serie. Ahora que ha comenzado el programa cargado en Arduino el siguiente paso aparecerá en un mensaje por pantalla con un botón de *OK*.

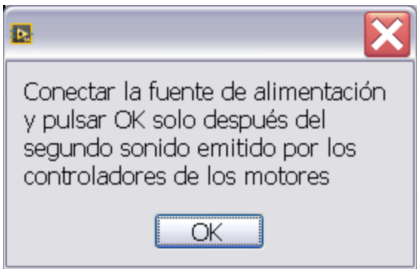


Figura 11.3: Botón OK.

La fuente de alimentación dispone de un interruptor para permitir/cortar el paso de la corriente. Como se indica en el mensaje de la figura 11.3 se debe pulsar el interruptor. Tras pulsar el interruptor se escuchará un sonido proveniente del ESC que indica que se ha encendido, después de ese primer sonido se escuchará otro sonido que indica que la programación del ESC ha entrado en modo *throttle*, que es en la que se quiere trabajar. Después de ese segundo sonido se debe pulsar el botón *OK*.

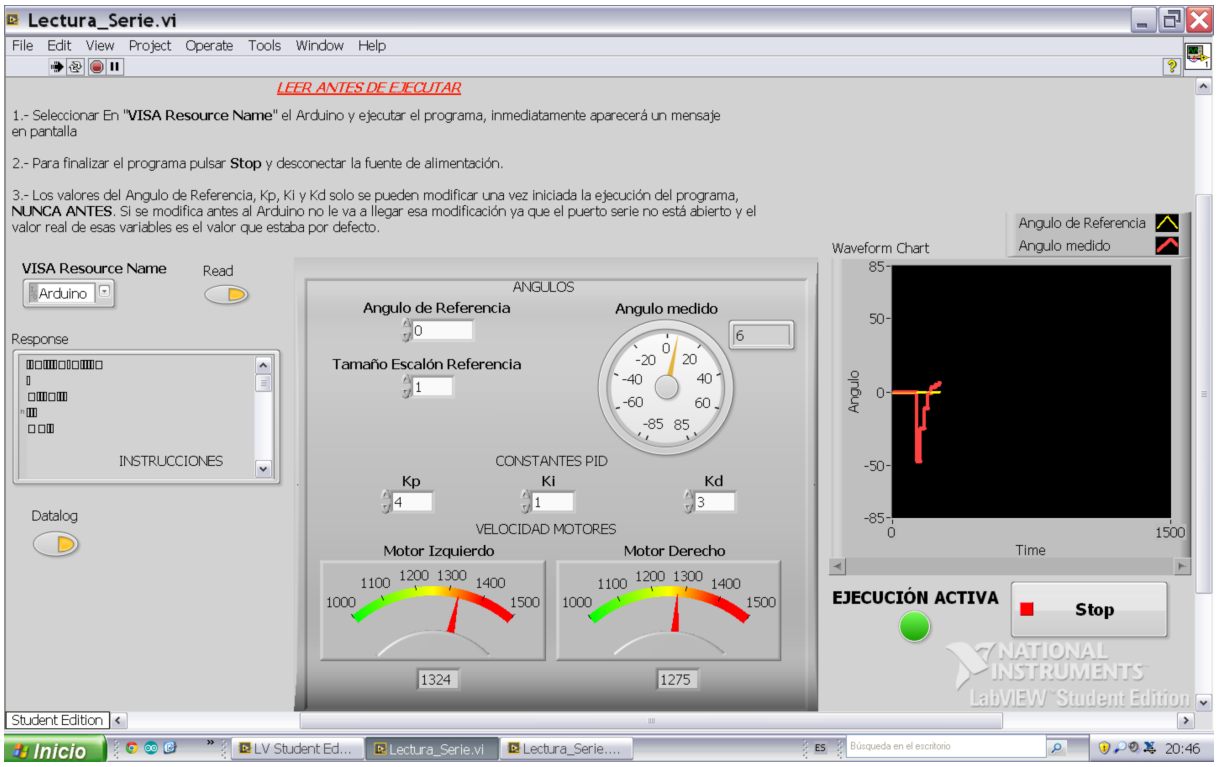


Figura 11.4: Ejecución activa.

La ejecución habrá comenzado, como indica la luz verde de la interfaz, y ya se puede manejar el brazo variando el ángulo de referencia y variando los valores de las constantes del controlador.

Capítulo 12

Bibliografía

[1] M. J. Cutler, *Design and control of an Autonomus Variable-Pitch Quadrotor Helicopter.* Massachussets Institute of Technology, 2012.

[2] K. LaFleur, K. Cassady, A. Doud, K. Shades, E. Rogin, and B. He, *Quadcopter control in three-dimensional space using a nonivasive motor imagery-based rai-computer interface.* Department of Biomedical Engineering, University of Minnesota, 2013.

[3] Pololu. Altimu-10. https://www.pololu.com/product/2470.

[4] Invensense. Mpu 6050. http://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/.

[5] M. Vila, *A quick overview on rotatory Brush and Brushless DC Motors.* Motion Control Department, Ingenia, 2006.

[6] Wikipedia. Giroscopo. https://es.wikipedia.org/wiki/Giroscopo.

[7] I. D. Hopkin, C. P. Fell, K. Townsend, and T. R. Mason, "Un giroscopio de estructura vibrante," http://www.oepm.es/pdf/ES/0000/000/02/23/29/ES-2232915_T3.pdf, Tech. Rep., 1998.

[8] STMicroelectronics. (2013) Hoja de caracteristicas giroscopio l3gd20h. http://www.st.com/web/catalog/sense_power/FM89/SC1288/PF254039.

[9] M. AndrejasÌic. (2008) Mems accelerometer. http://www.st.com/web/catalog/sense_power/FM89/SC1288/PF254039.

[10] STMicroelectronics. (2013) Hoja de caracteristicas acelerometro lsm303d. http://www.st.com/web/en/resource/technical/document/datasheet/DM00057547.pdf.

[11] F. Semiconductor. (2013) Tilt sensing using a three axis accelerometer. http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf.

[12] Fritzing. Esquemas de circuitos. http://fritzing.org/home/.

[13] R. circuits. How to destroy an arduino (method 5). http://www.ruggedcircuits.com/10-ways-to-destroy-an-arduino.

[14] O. Katsuhiko, *Ingenieria de control moderna.* University of Minnesota, 1998.

[15] D. Zhang, H. Li, and E. G. Collins, *Digital anti-windup PI controllers for variable-speed motor drives using FPGA and stochastic theory.* IEEE TRANSACTIONS ON POWER ELECTRONICS, 2006.

[16] N. Instruments. Labview. http://spain.ni.com.

[17] NI, "Comunicacion serial utilizando labview con un microcontrolador," http://www.ni.com/white-paper/7907/es/, Tech. Rep., 2008.

[18] Arduino. Ide arduino. https://www.arduino.cc/en/Main/Software.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá